Sequence Abstractions for Flexible, Line-Rate Network Monitoring

Andrew Johnson[1], Ryan Beckett[2], Xiaoqi Chen[13], Ratul Mahajan[4], and David Walker[1]

NSDI 2024

April 18, 2024

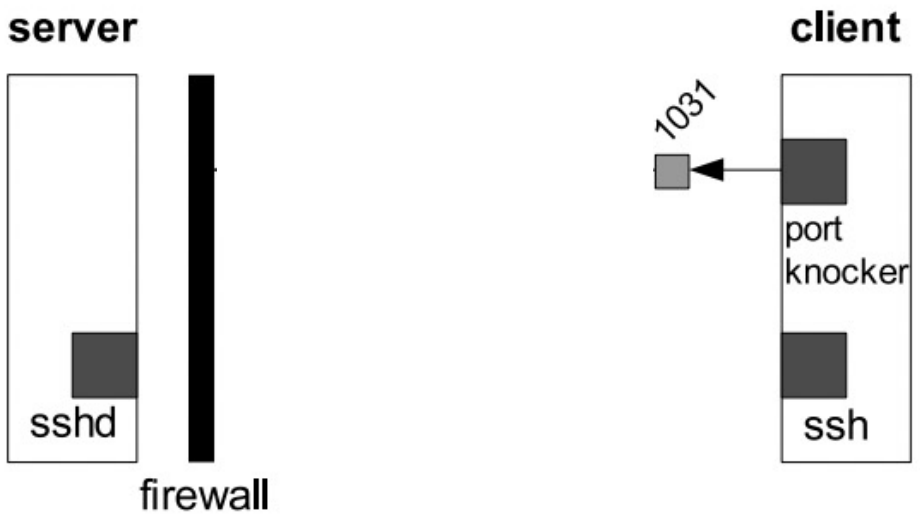
1: Princeton University

2: Microsoft Research

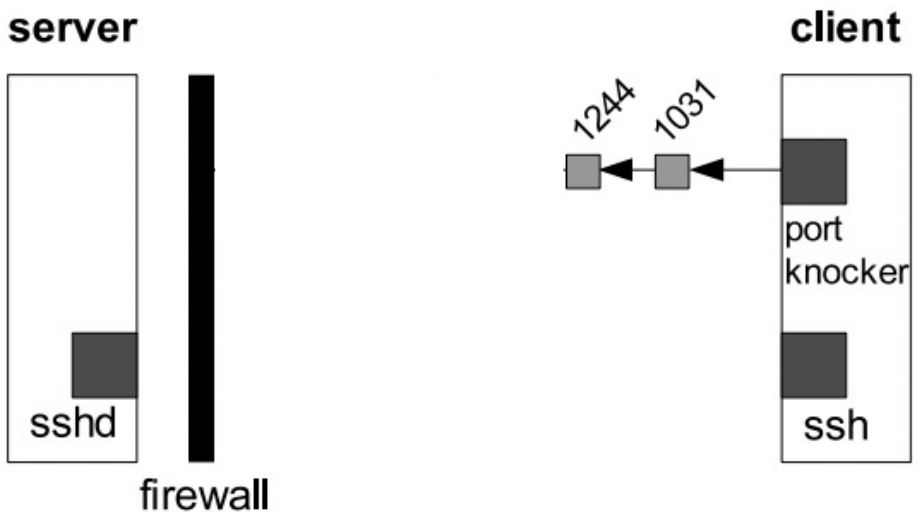
3: Purdue University

4: University of Washington

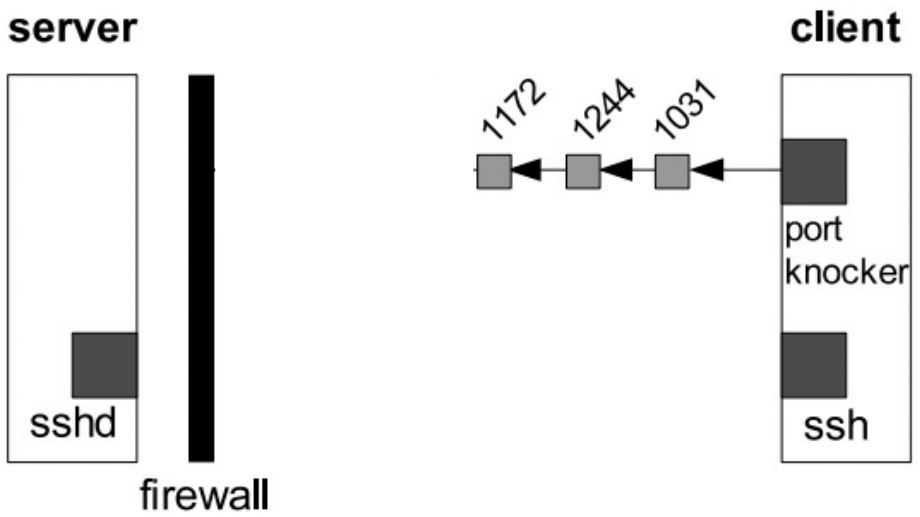
In Port Knocking (deGraaf 2005)



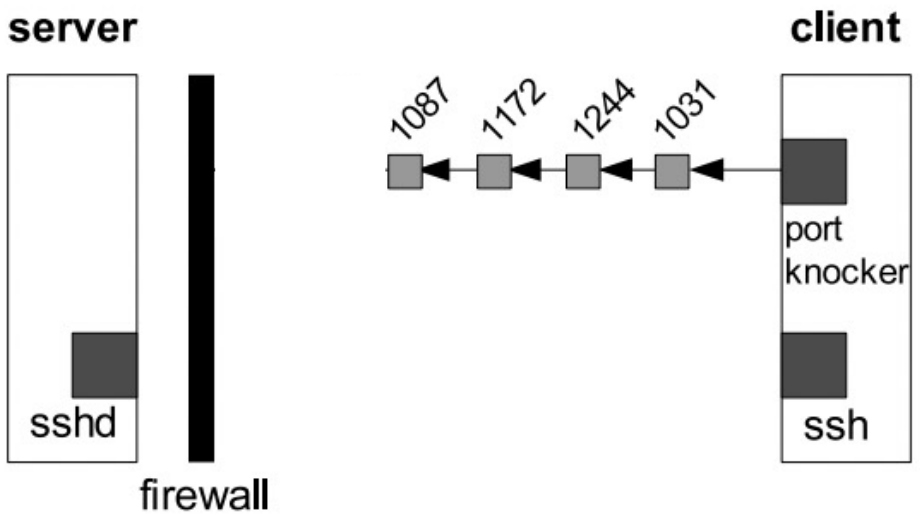
In Port Knocking (deGraaf 2005)



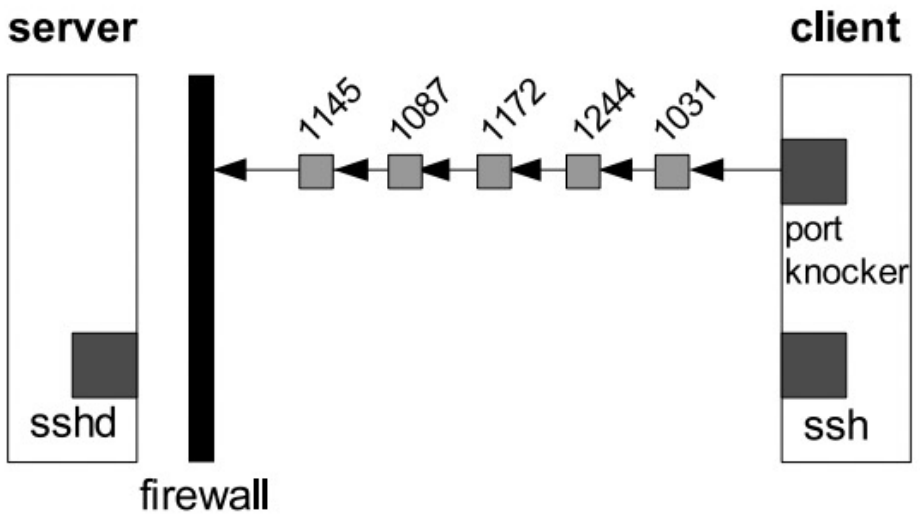
In Port Knocking (deGraaf 2005)



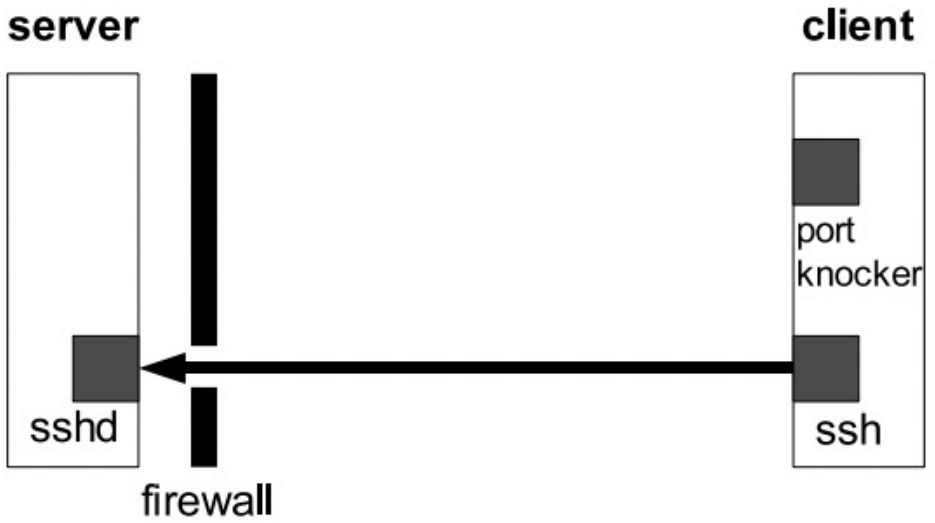
In Port Knocking (deGraaf 2005)



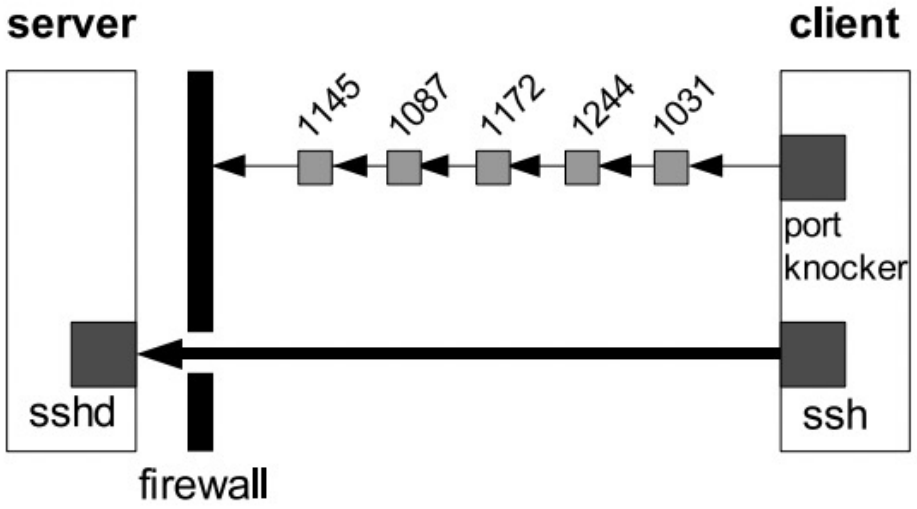
In Port Knocking (deGraaf 2005)



In Port Knocking (deGraaf 2005)

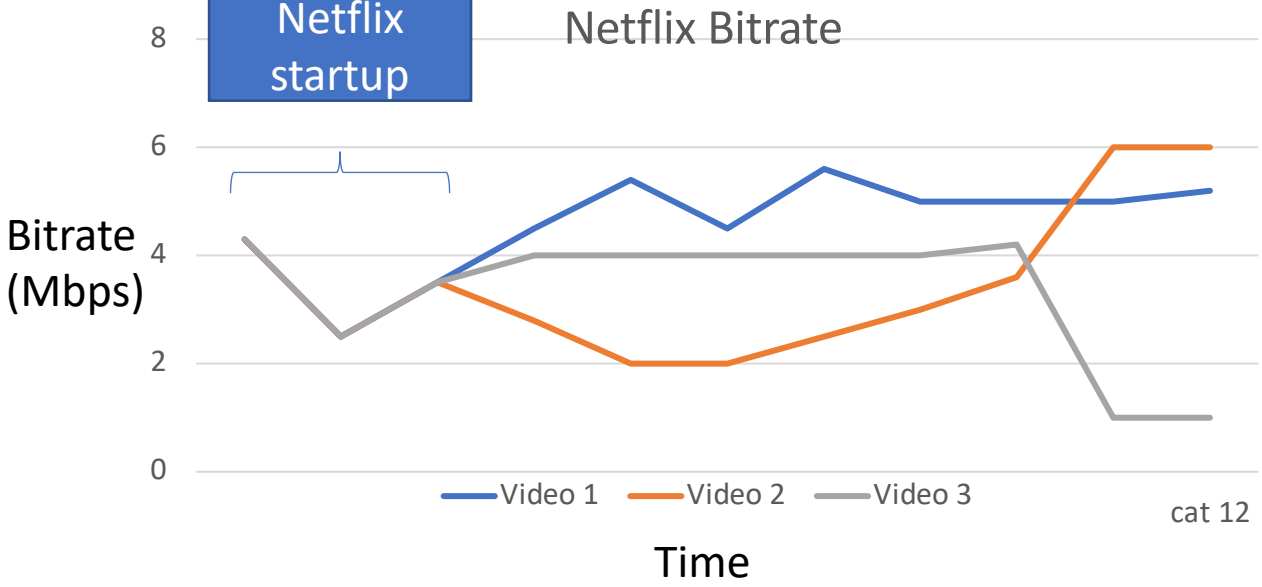


In Port Knocking (deGraaf 2005)

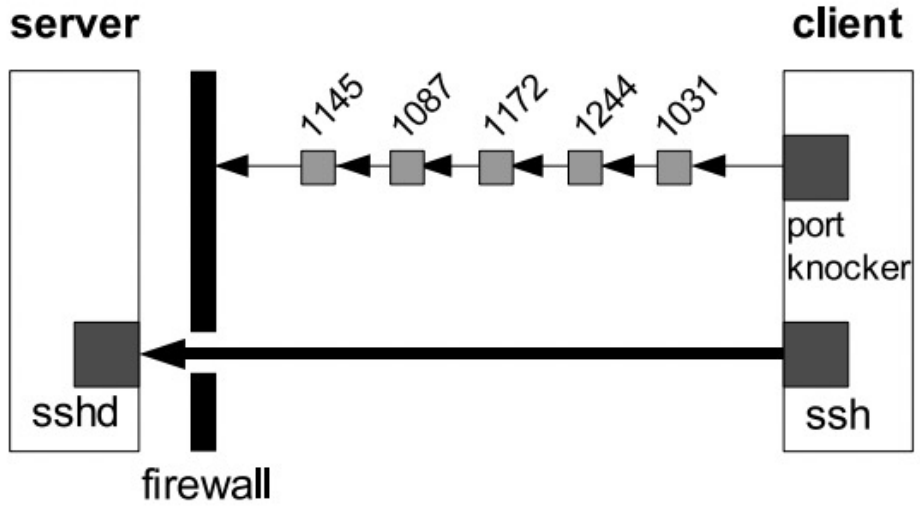


In Video Fingerprinting (Schuster 2017)

Netflix startup

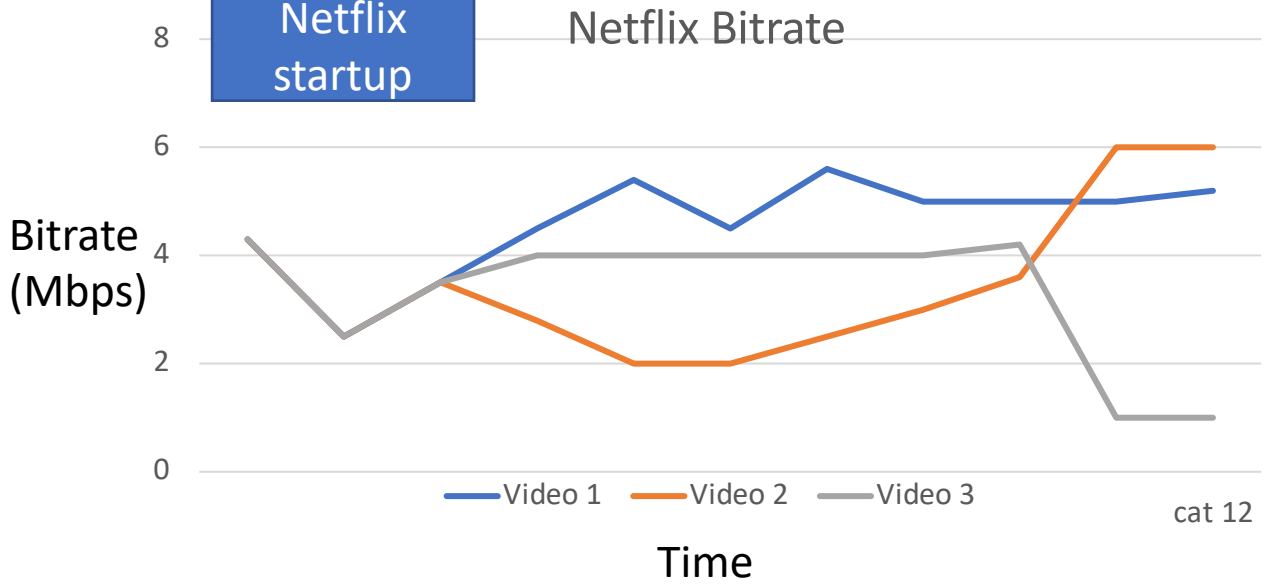


In Port Knocking (deGraaf 2005)

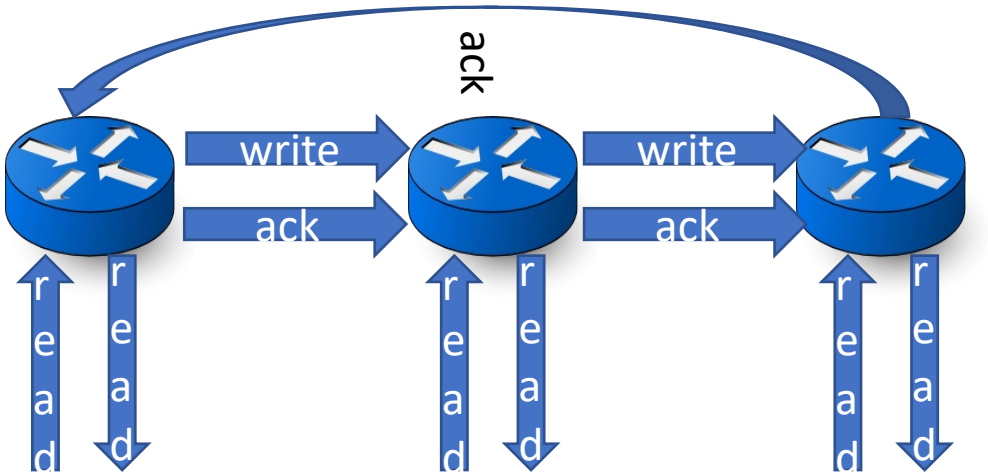


In Video Fingerprinting (Schuster 2017)

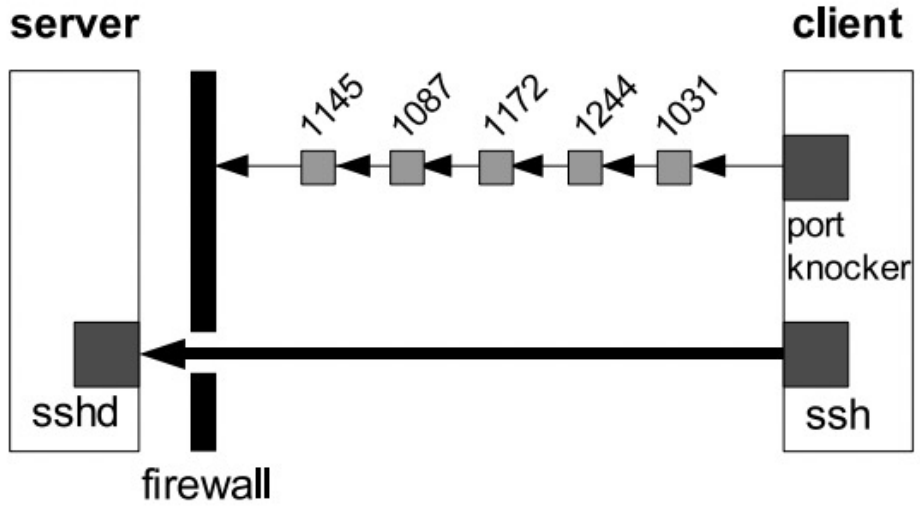
Netflix startup



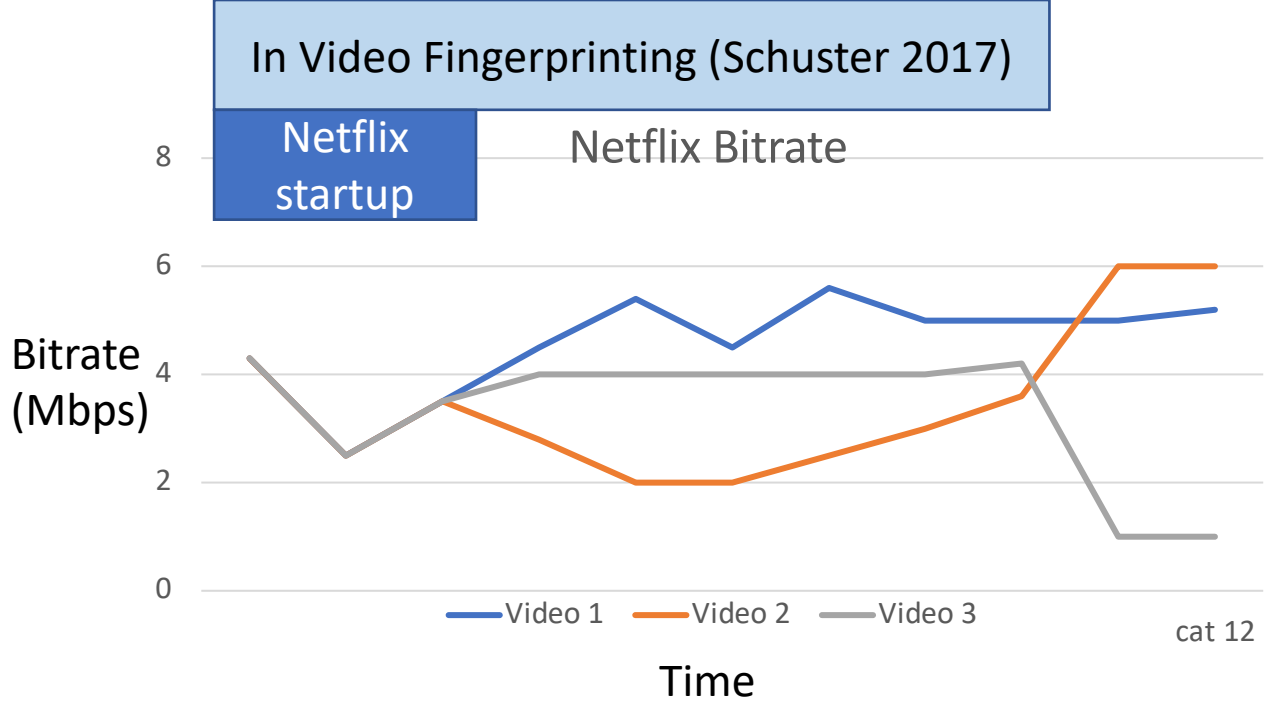
In Network Applications (Zeno 2022)



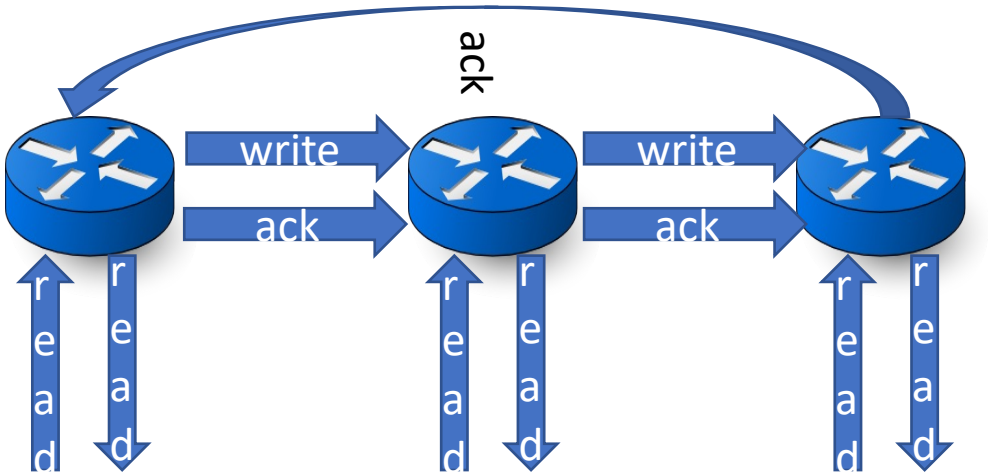
In Port Knocking (deGraaf 2005)



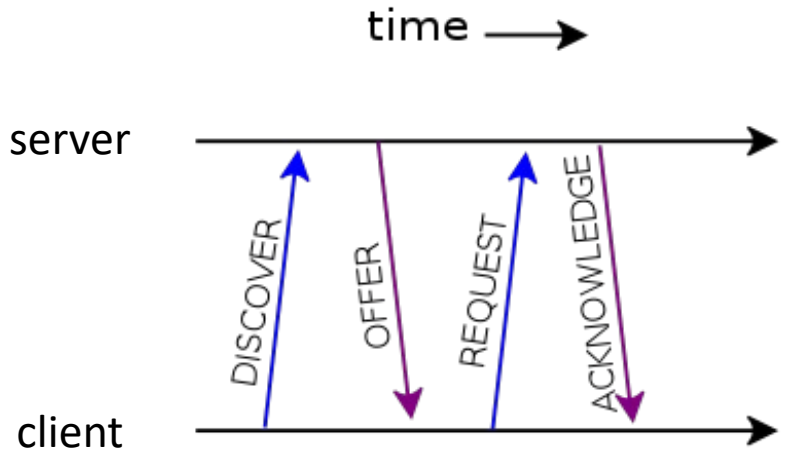
In Video Fingerprinting (Schuster 2017)



In Network Applications (Zeno 2022)

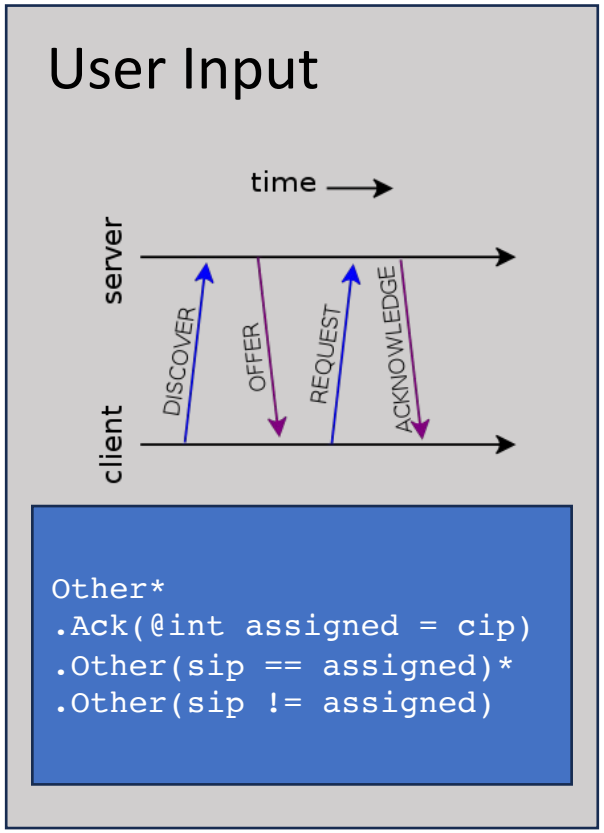


In Network Protocols (DHCP, TCP)



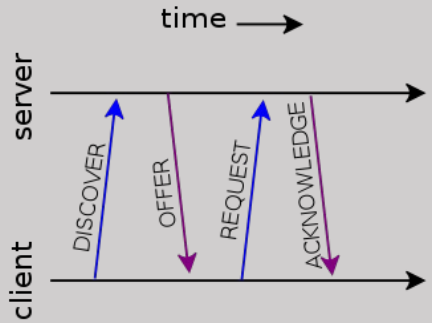
FLM Overview

FLM Overview



FLM Overview

User Input

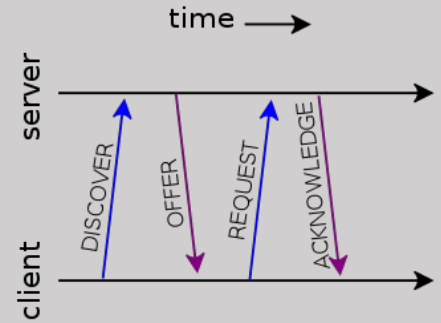



```
Other*
.Ack(@int assigned = cip)
.Other(sip == assigned)*
.Other(sip != assigned)
```

Compiler

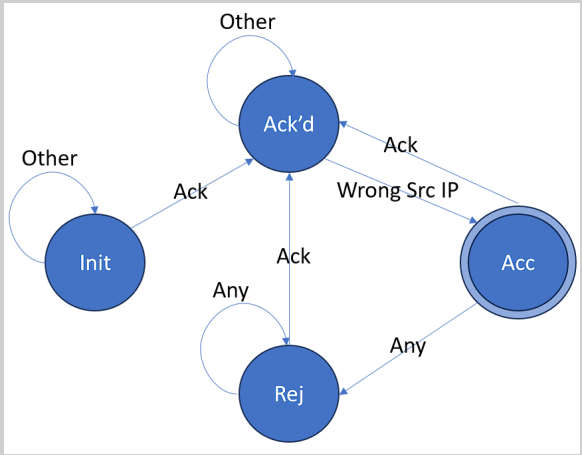
FLM Overview

User Input



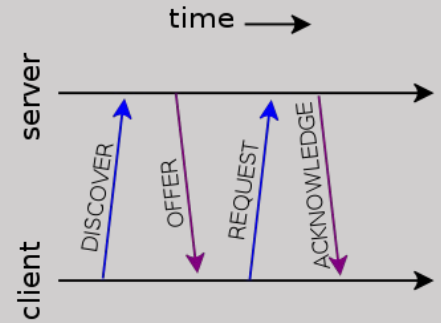

```
Other*
.Ack(@int assigned = cip)
.Other(sip == assigned)*
.Other(sip != assigned)
```

Compiler



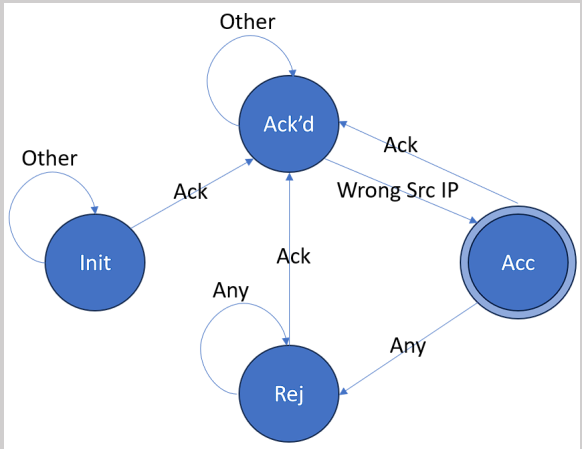
FLM Overview

User Input




```
Other*
.Ack(@int assigned = cip)
.Other(sip == assigned)*
.Other(sip != assigned)
```

Compiler

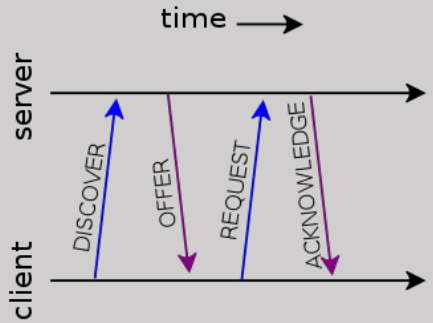



```
transition(curr, input):
  if (curr + input > 9):
    return input xor 5
  else:
    return curr & 1
```

FLM Overview

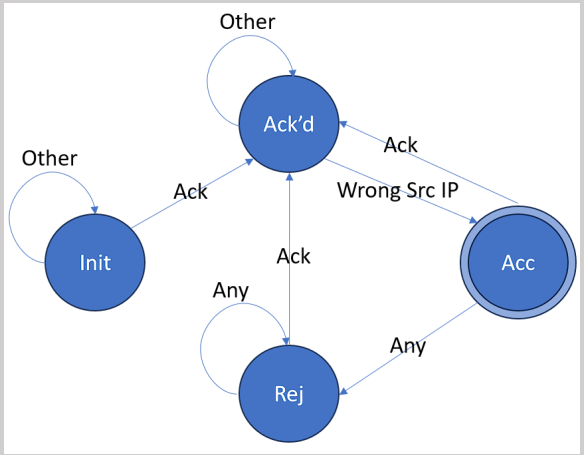
- Definition of pattern language syntax and semantics
- Provably correct compiler from patterns to P4
- Evaluation on 15 different networking applications

User Input




```
Other*
.Ack(@int assigned = cip)
.Other(sip == assigned)*
.Other(sip != assigned)
```

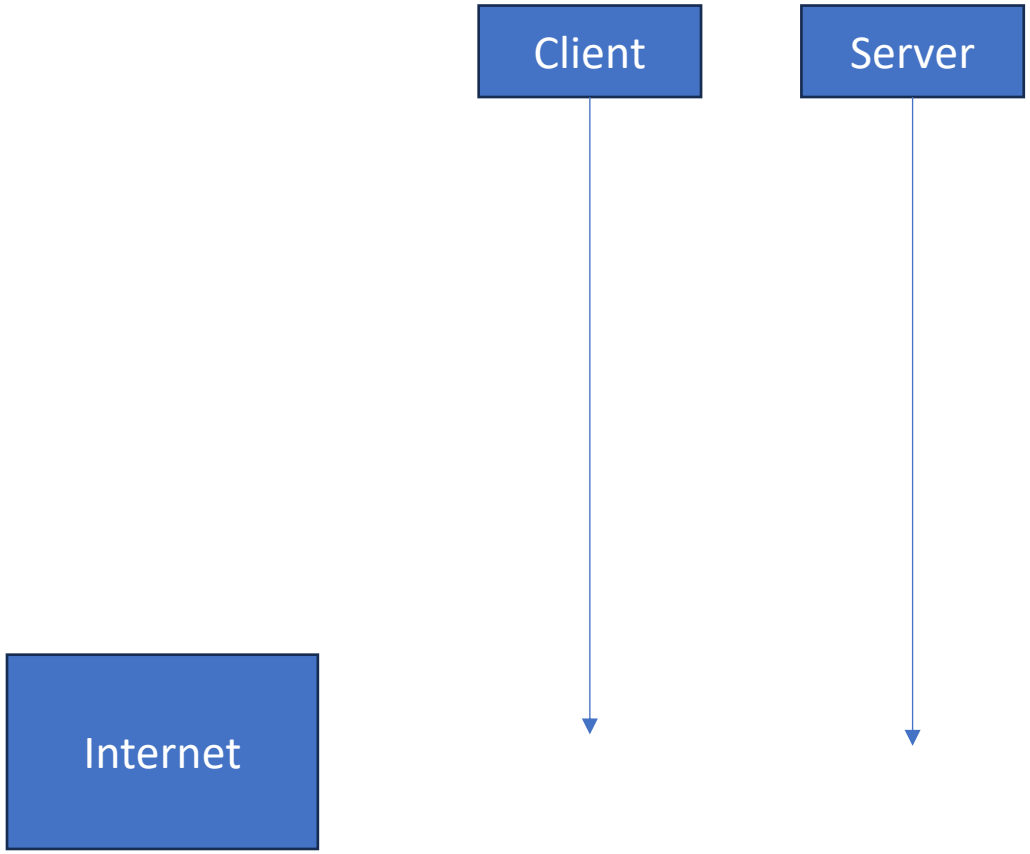
Compiler



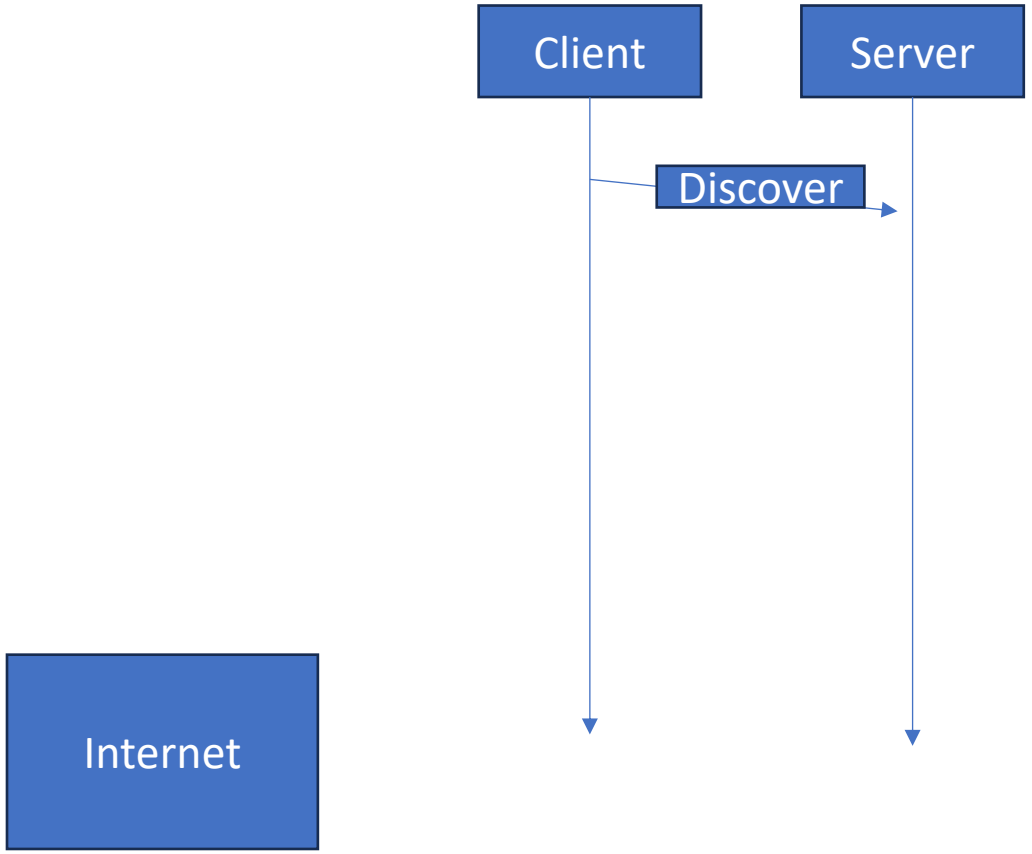

```
transition(curr, input):
  if (curr + input > 9):
    return input xor 5
  else:
    return curr & 1
```

Example FLM Program: Detecting DHCP Anomaly

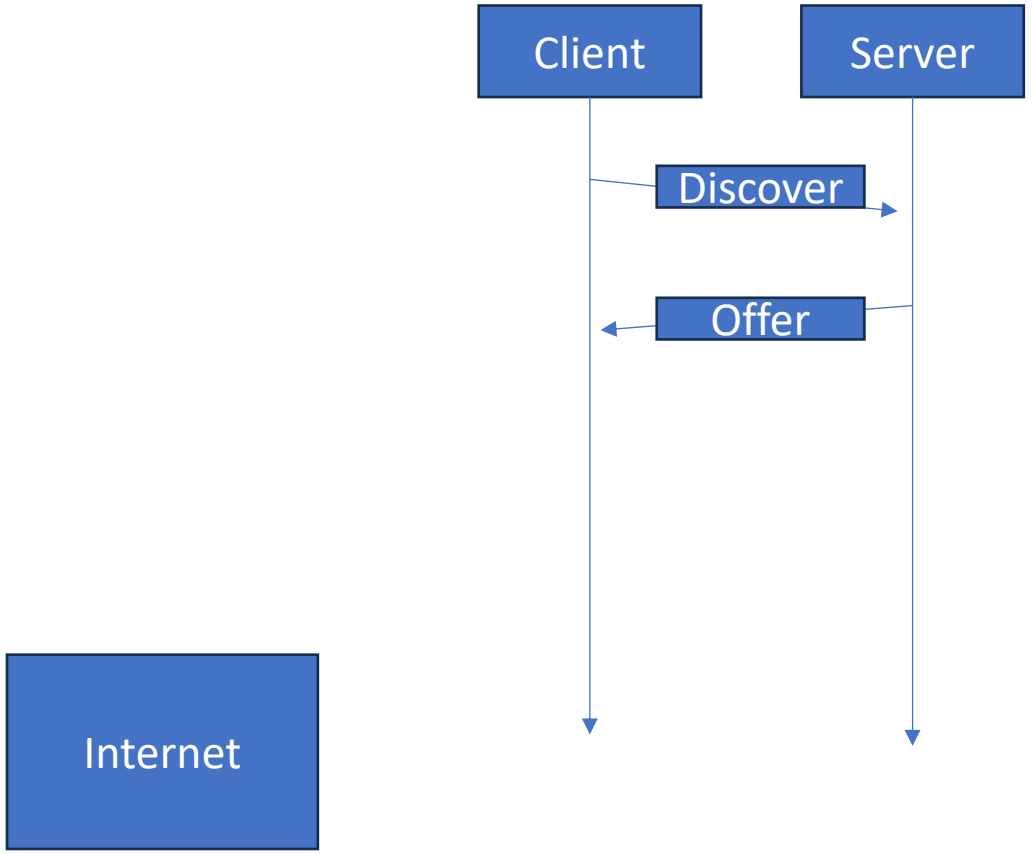
DHCP via FLM: Events



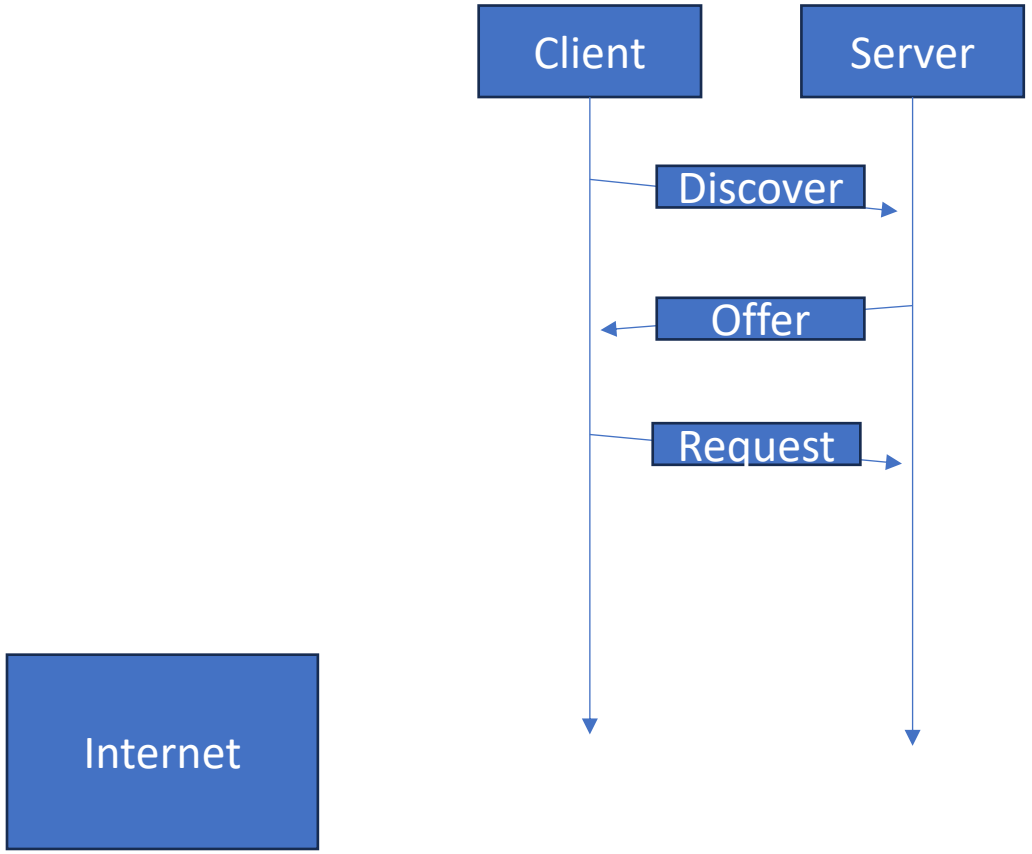
DHCP via FLM: Events



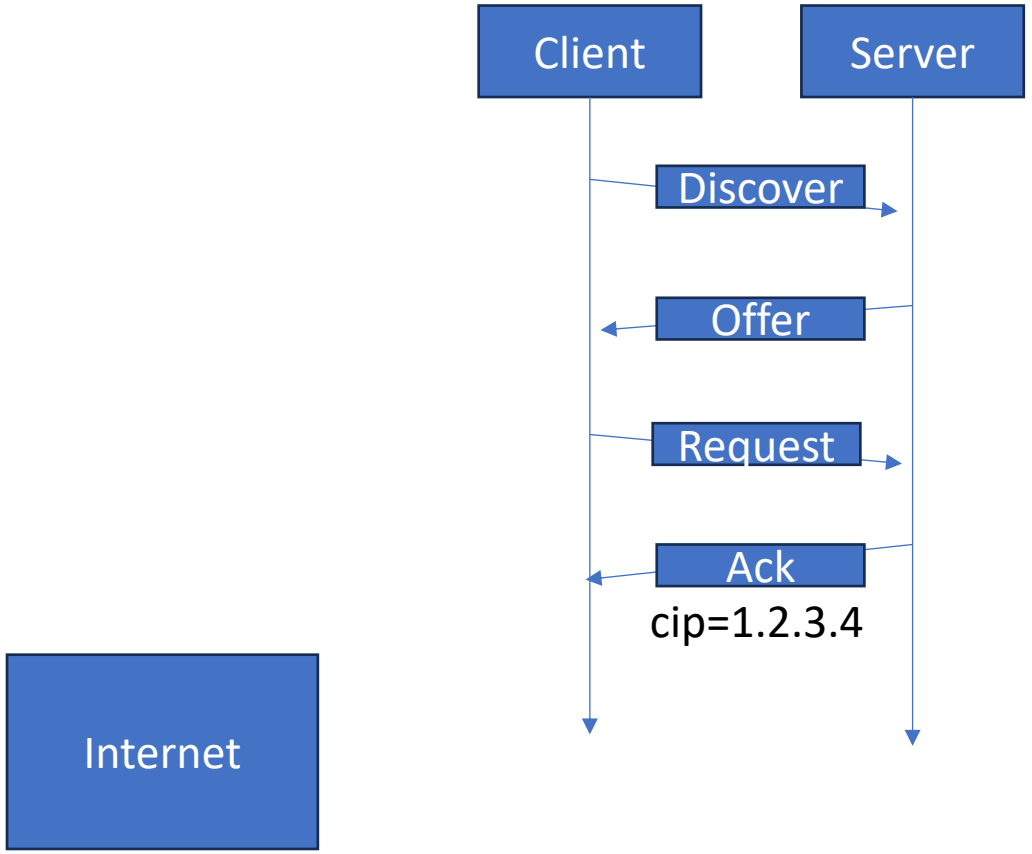
DHCP via FLM: Events



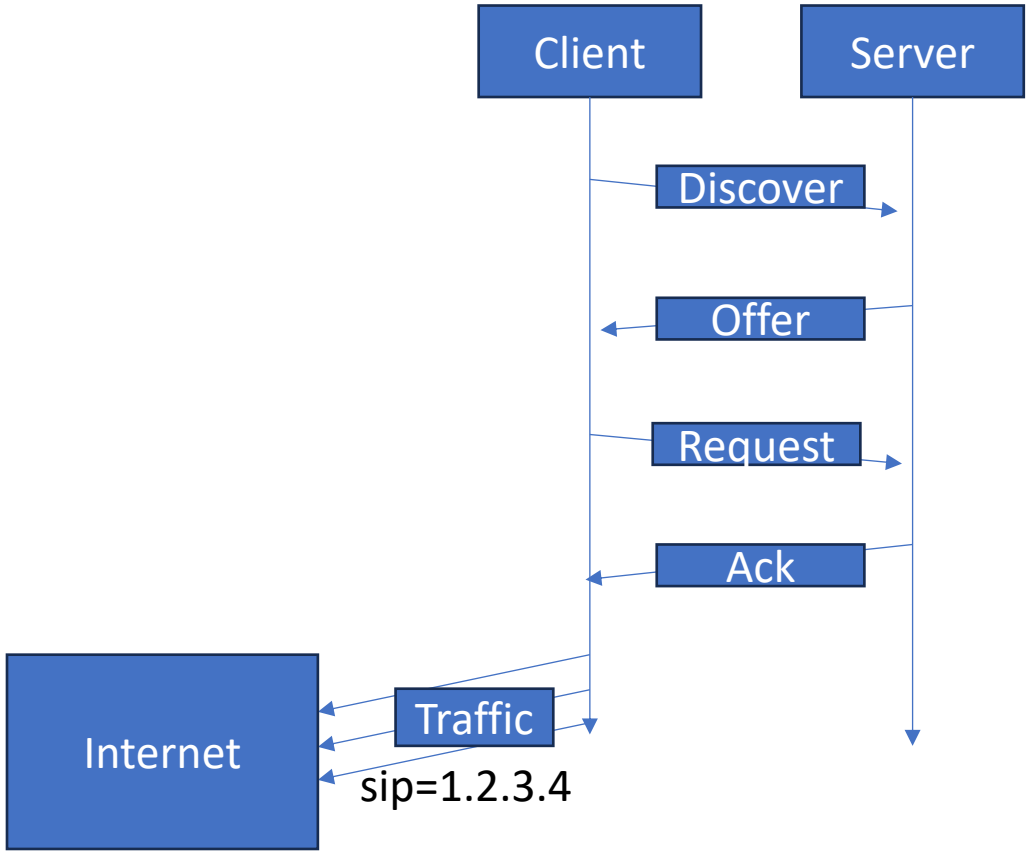
DHCP via FLM: Events



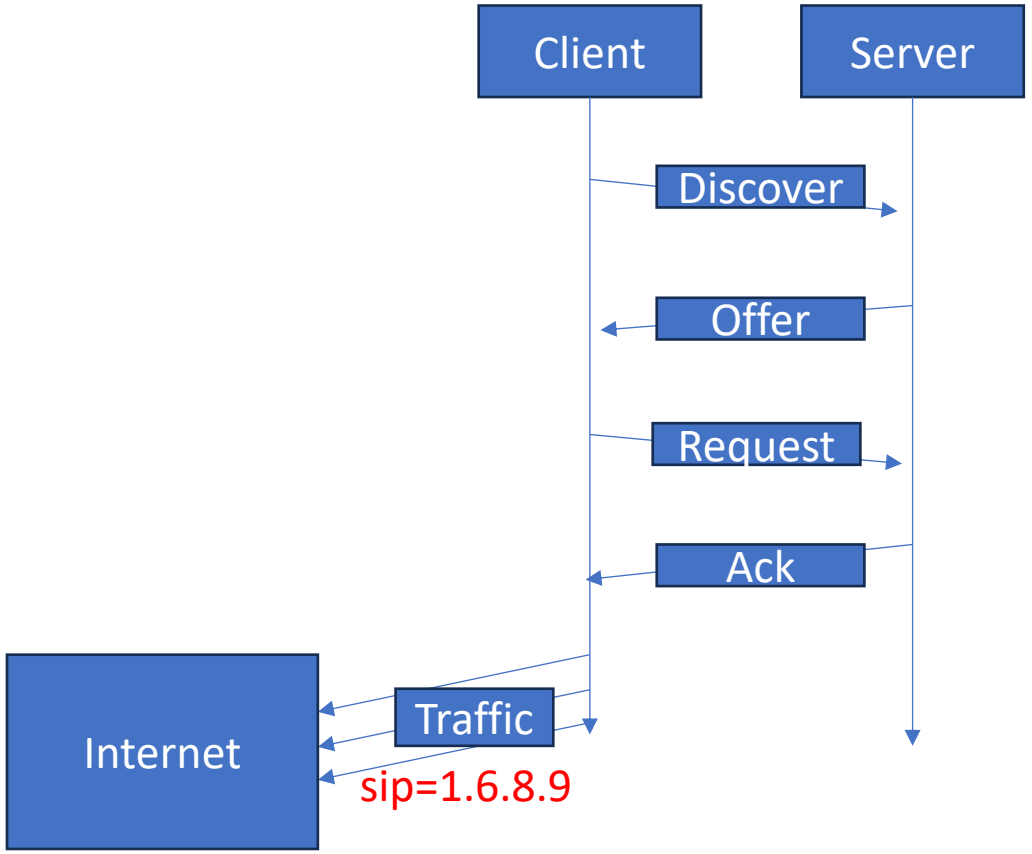
DHCP via FLM: Events



DHCP via FLM: Events



DHCP via FLM: Events

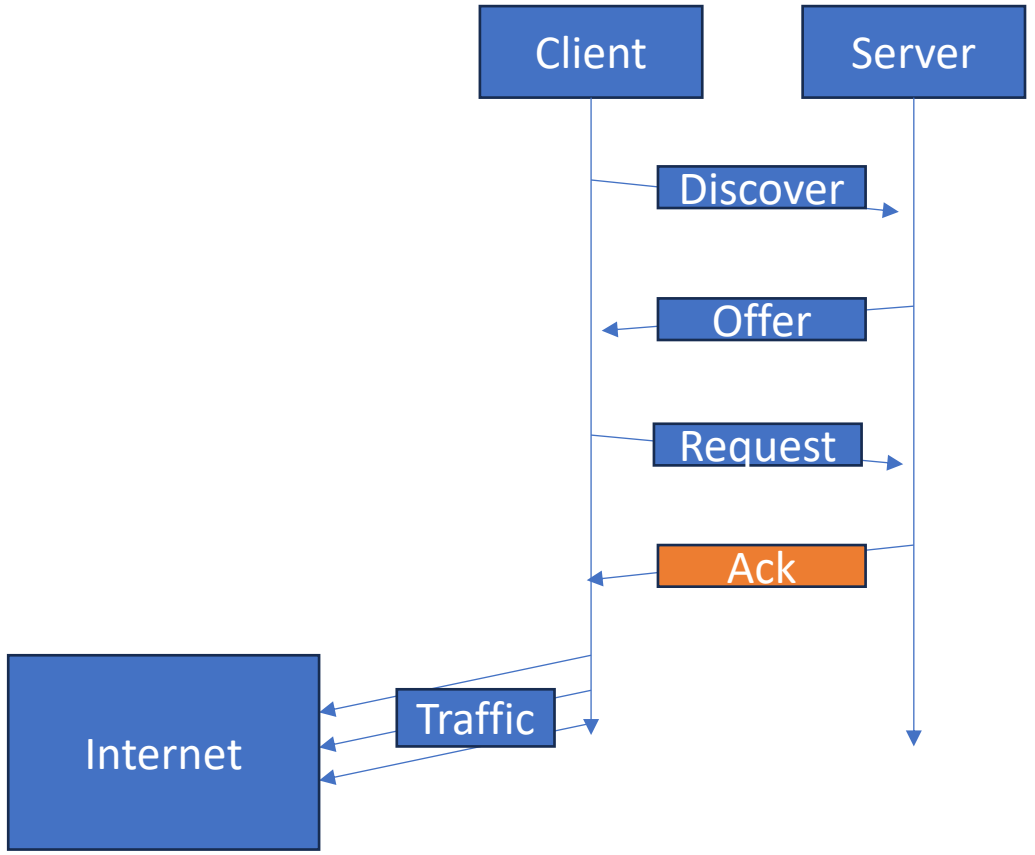


DHCP via FLM: Events

- The ACKNOWLEDGE message

  `event Ack(int cip, int cmac)`

  - `cip:` assigned client IP
  - `cmac:` client MAC (ID)



DHCP via FLM: Events

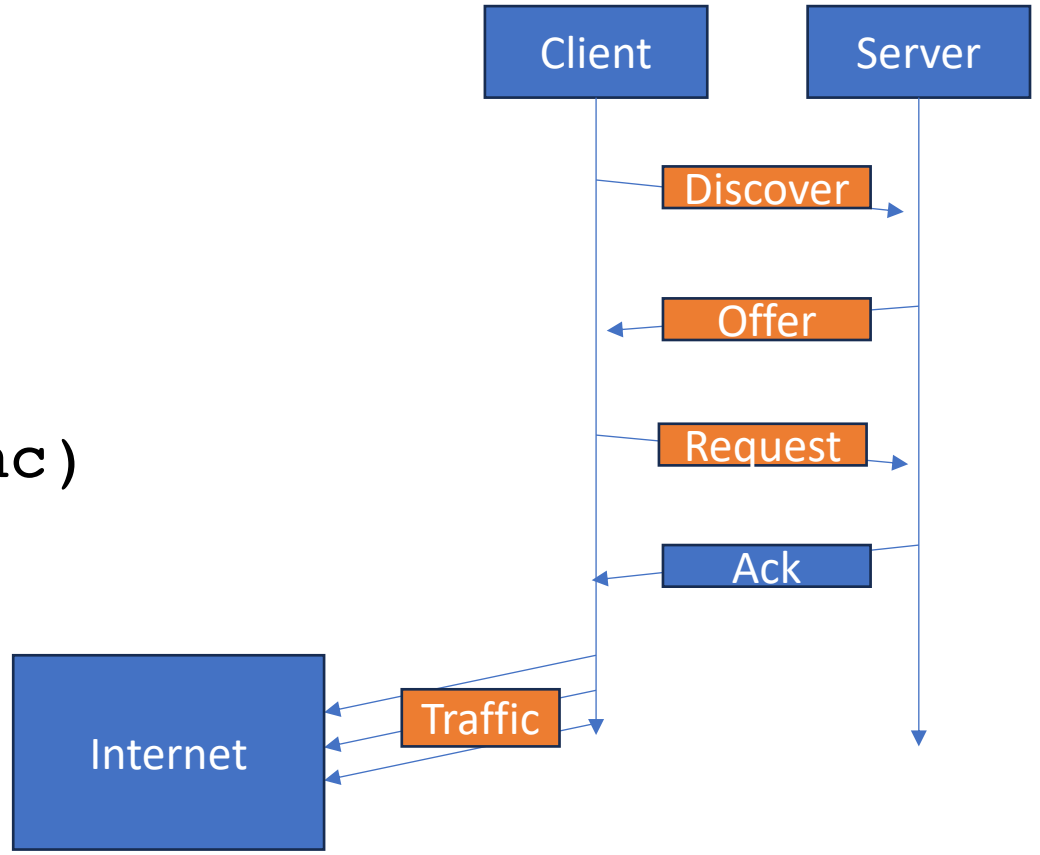
- The ACKNOWLEDGE message

  `event Ack(int cip, int cmac)`

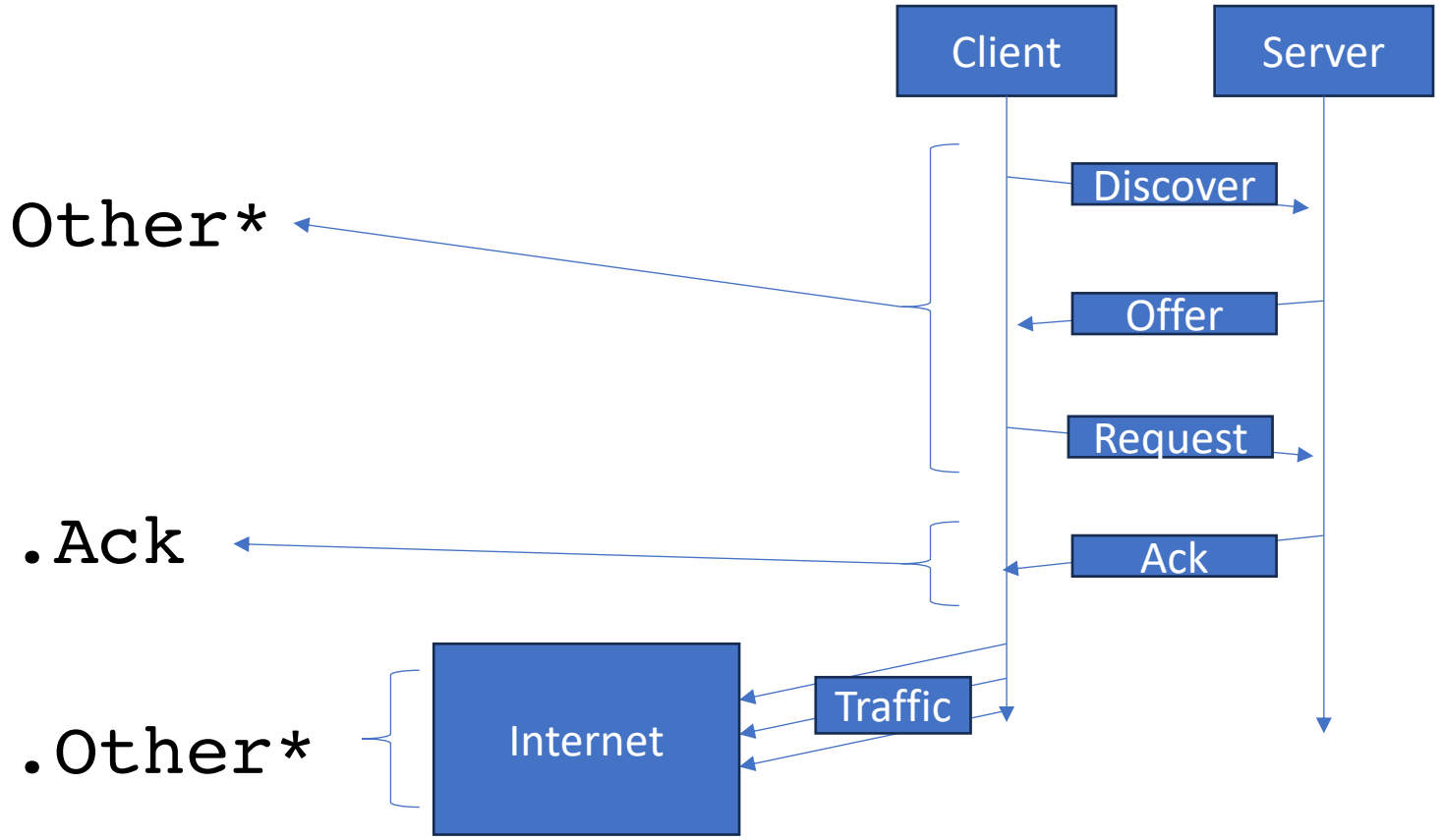
  - `cip:` assigned client IP
  - `cmac:` client MAC (ID)
- All other messages

  `event Other(int sip, int smac)`

  - `sip:` source IP
  - `smac:` source MAC (ID)



DHCP via FLM: Basic Pattern

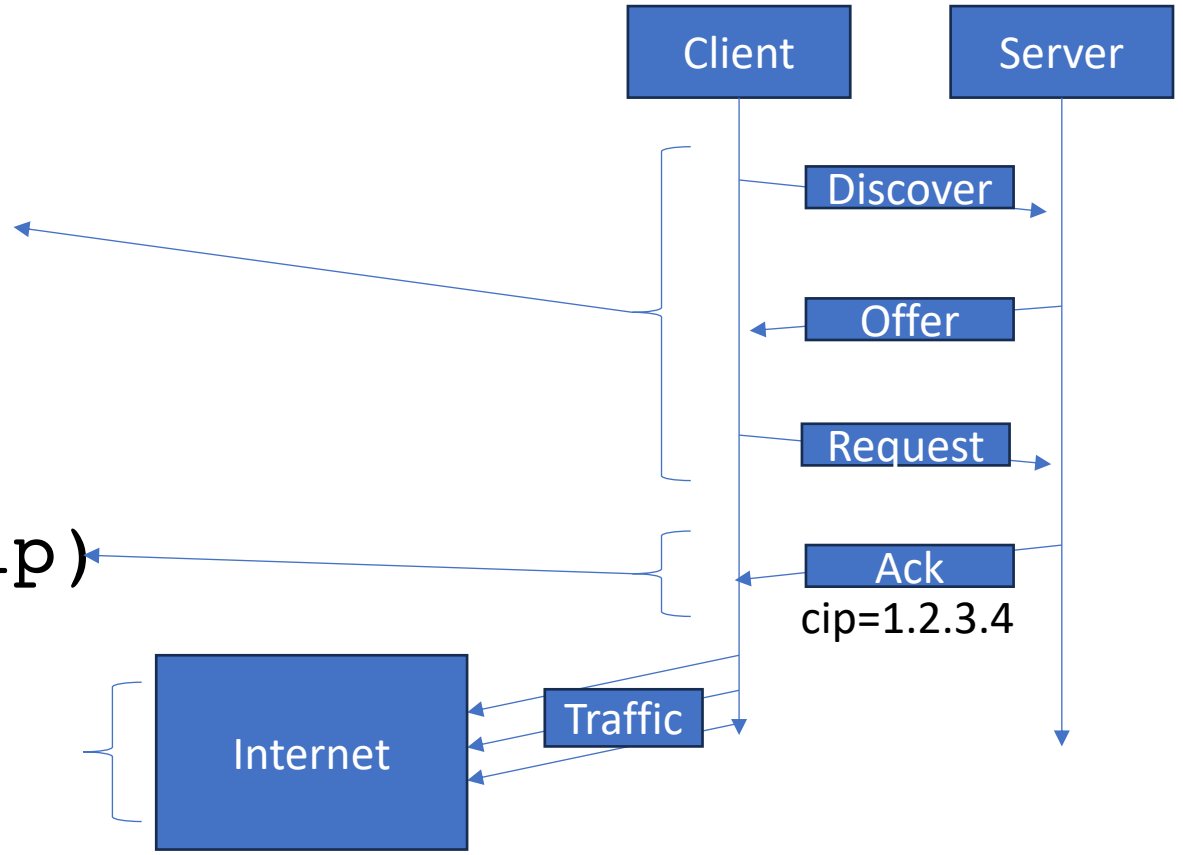


DHCP via FLM: Bindings

```
Other*

.Ack(@int assigned = cip)

.Other*
```

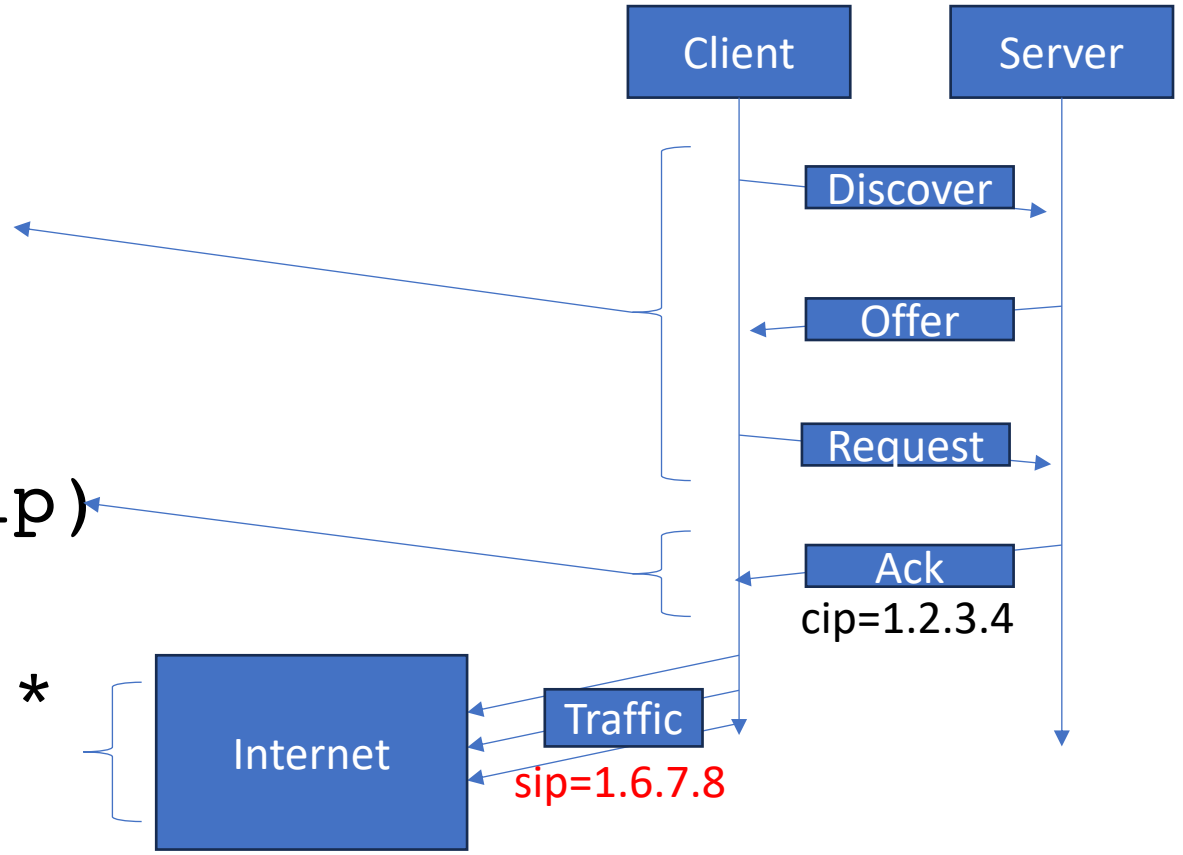


DHCP via FLM: Predicates

```
Other*

.Ack(@int assigned = cip)

.Other(sip == assigned)*
.Other(sip != assigned)
```



DHCP via FLM: Final Program

```
event Ack(int cip, int cmac);
event Other(int sip, int smac);
spec<2048> dhcp_misuse =
IDX {
    Ack -> return Hash(cmac)
    Other -> return Hash(smac)
}
DETECT {
     Other*
    .Ack(@int assigned = cip)
    .Other(sip == assigned)*
    .Other(sip != assigned)
} => {
    (…some response…)
}
```

DHCP via FLM: Final Program

```
event Ack(int cip, int cmac);
event Other(int sip, int smac);
spec<2048> dhcp_misuse =
IDX {
    Ack -> return Hash(cmac)
    Other -> return Hash(smac)
}
DETECT {
     Other*
    .Ack(@int assigned = cip)
    .Other(sip == assigned)*
    .Other(sip != assigned)
} => {
    (…some response…)
}
```

Declares Event Types

DHCP via FLM: Final Program

```
event Ack(int cip, int cmac);
event Other(int sip, int smac);
spec<2048> dhcp_misuse =
IDX {
    Ack -> return Hash(cmac)
    Other -> return Hash(smac)
}
DETECT {
     Other*
    .Ack(@int assigned = cip)
    .Other(sip == assigned)*
    .Other(sip != assigned)
} => {
    (…some response…)
}
```

Declares Event Types

Generates Flow Identifier

DHCP via FLM: Final Program

```
event Ack(int cip, int cmac);
event Other(int sip, int smac);
spec<2048> dhcp_misuse =
IDX {
    Ack -> return Hash(cmac)
    Other -> return Hash(smac)
}
DETECT {
     Other*
    .Ack(@int assigned = cip)
    .Other(sip == assigned)*
    .Other(sip != assigned)
} => {
    (…some response…)
}
```

Declares Event Types

Generates Flow Identifier

Specifies Error Sequence

DHCP via FLM: Final Program

```
event Ack(int cip, int cmac);
event Other(int sip, int smac);
spec<2048> dhcp_misuse =
IDX {
    Ack -> return Hash(cmac)
    Other -> return Hash(smac)
}
DETECT {
     Other*
    .Ack(@int assigned = cip)
    .Other(sip == assigned)*
    .Other(sip != assigned)
} => {
    (…some response…)
}
```

Declares Event Types

Generates Flow Identifier

Specifies Error Sequence

Defines Action On Match

Compilation of FLM

```
 Other*
.Ack(@int assigned = cip)
.Other(sip == assigned)*
.Other(sip != assigned)
```

```
 Other*
.Ack(@int assigned = cip)
.Other(sip == assigned)*
.Other(sip != assigned)
```

Remove Bindings

On Ack:
assigned := cip

```
 Other*
.Ack
.Other(sip == assigned)*
.Other(sip != assigned)
```

```
 Other*
.Ack(@int assigned = cip)
.Other(sip == assigned)*
.Other(sip != assigned)
```

Remove Bindings

On Ack:
assigned := cip

```
 Other*
.Ack
.Other(sip == assigned)*
.Other(sip != assigned)
```

Remove Prediactes

On Ack:
assigned := cip

```
 (Other1 + Other0)*
.Ack
.Other1*
.Other0
```

```
 Other*
.Ack(@int assigned = cip)
.Other(sip == assigned)*
.Other(sip != assigned)
```

Remove Bindings

On Ack:
assigned := cip

```
 Other*
.Ack
.Other(sip == assigned)*
.Other(sip != assigned)
```

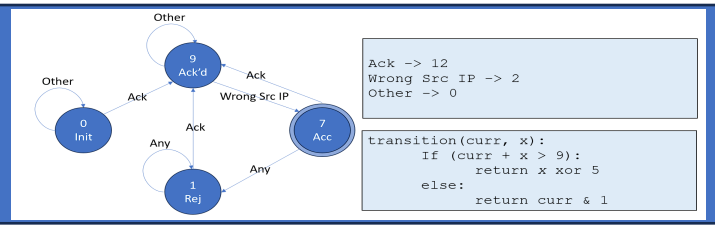
Remove Prediactes

On Ack:
assigned := cip

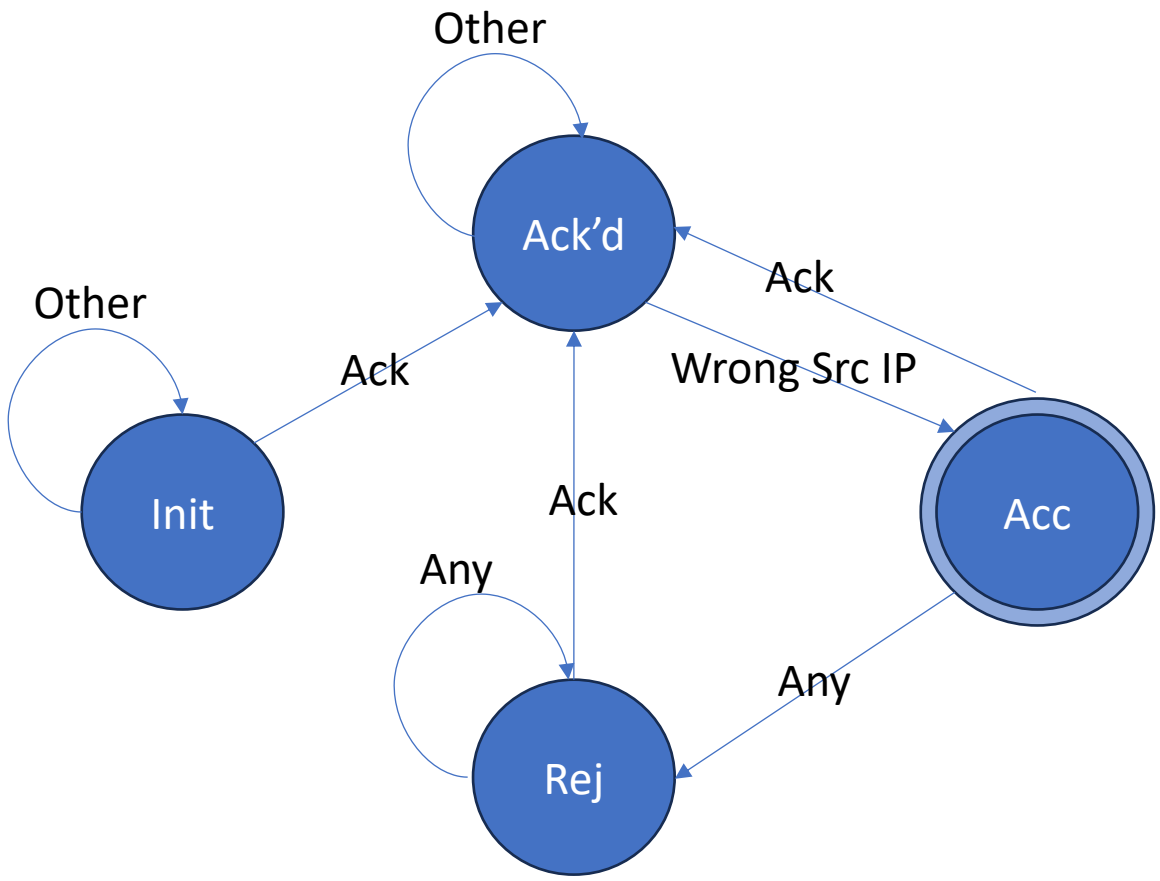
```
 (Other1 + Other0)*
.Ack
.Other1*
.Other0
```

Synthesize DFA

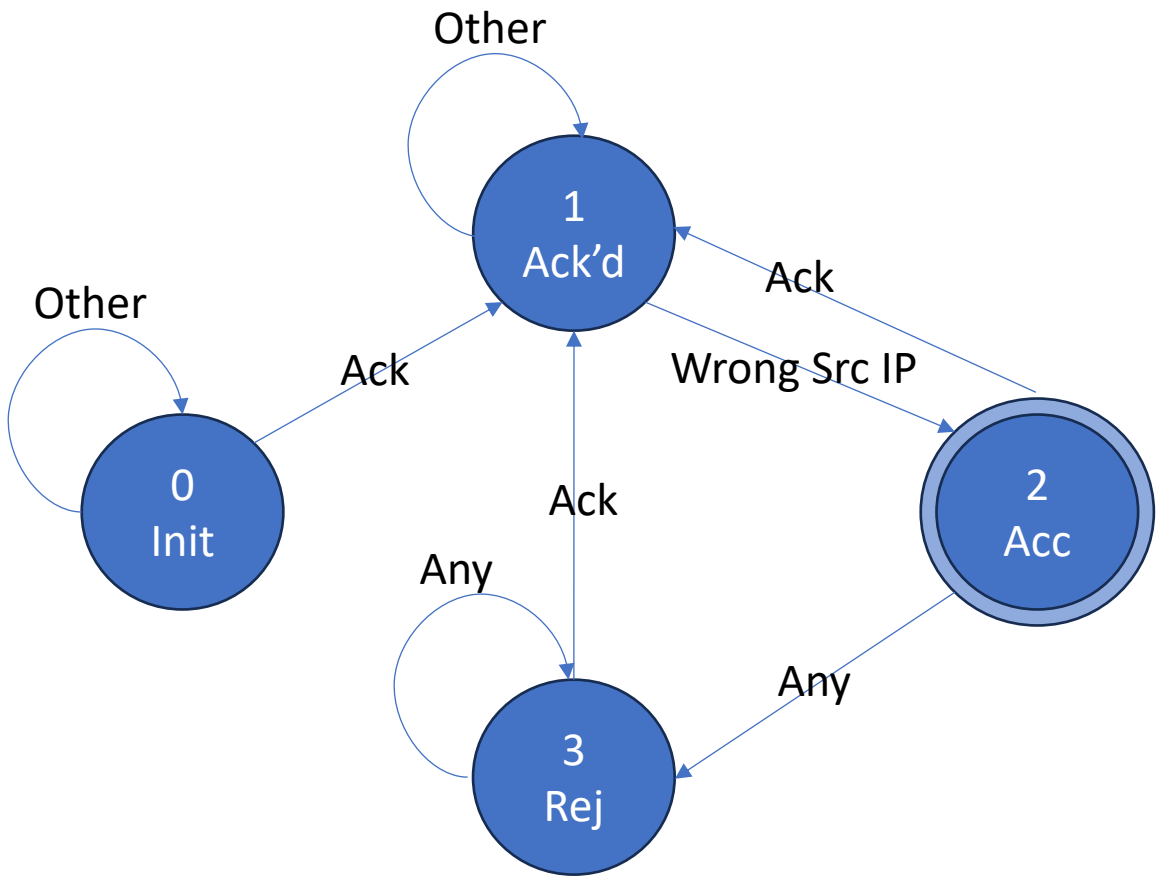
On Ack:
assigned := cip



Embedding State Machines in Hardware



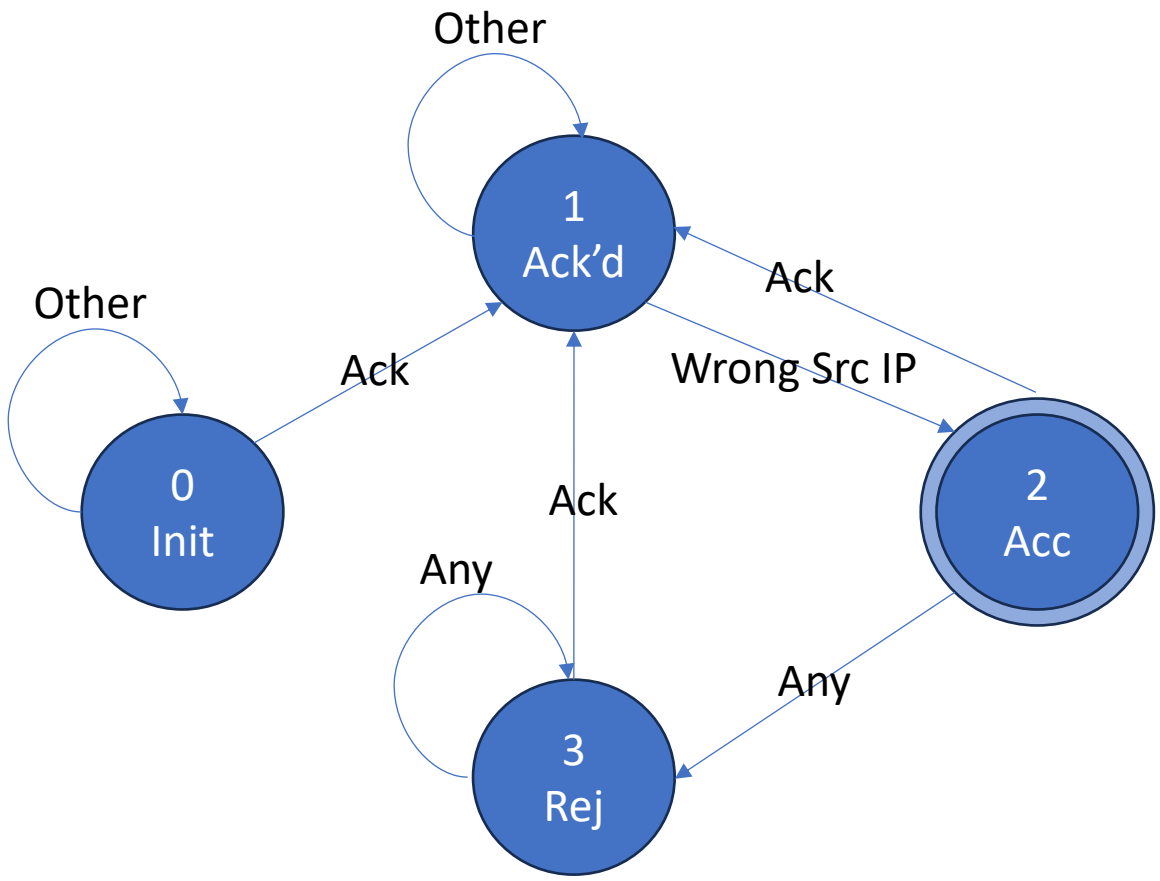
Embedding State Machines in Hardware




```
Ack -> 0
Wrong Src IP -> 1
Other -> 2
```

```
transition(state, input):
  if(state==0 & input==0):
    state := 1
  elif(state-=1 & input==0):
    state := 3
  elif(state==3 & input==0):
    state := 1
. . .
```

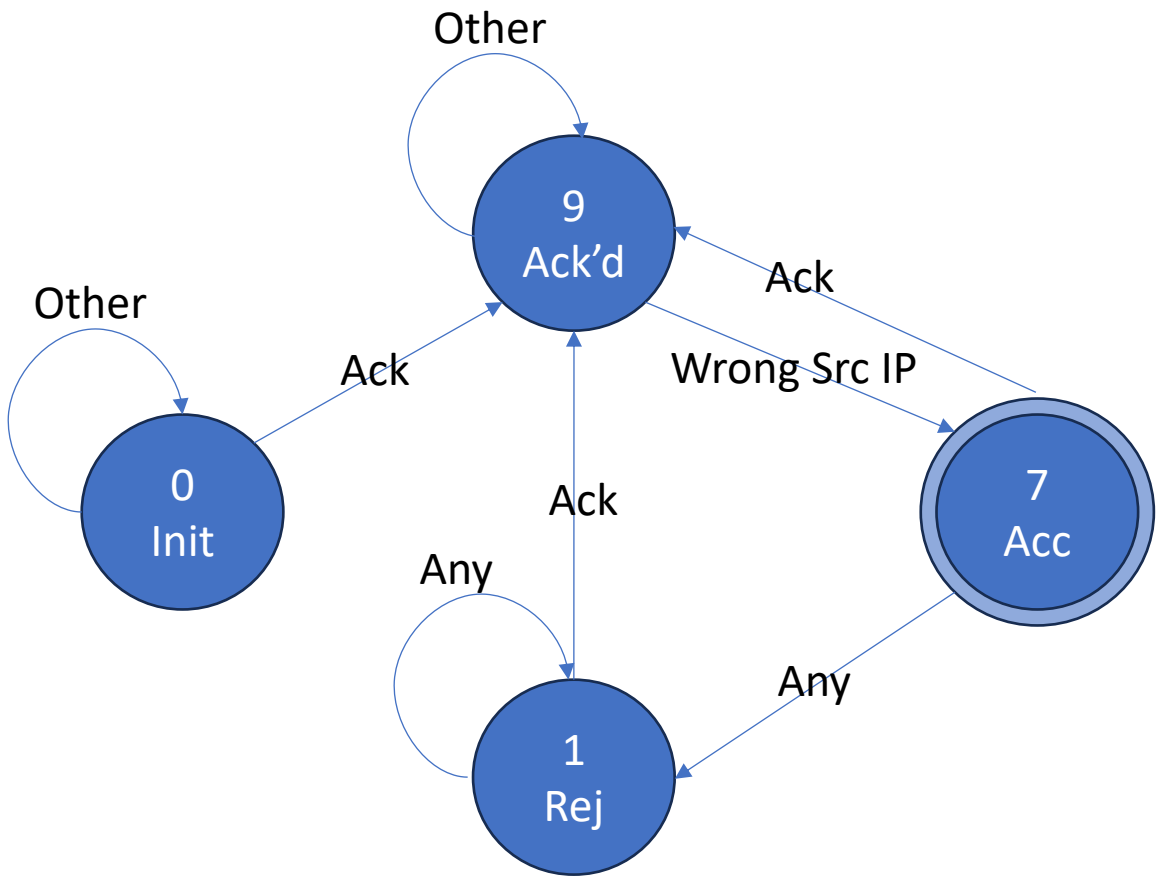
Embedding State Machines in Hardware




```
Ack -> 0
Wrong Src IP -> 1
Other -> 2
```

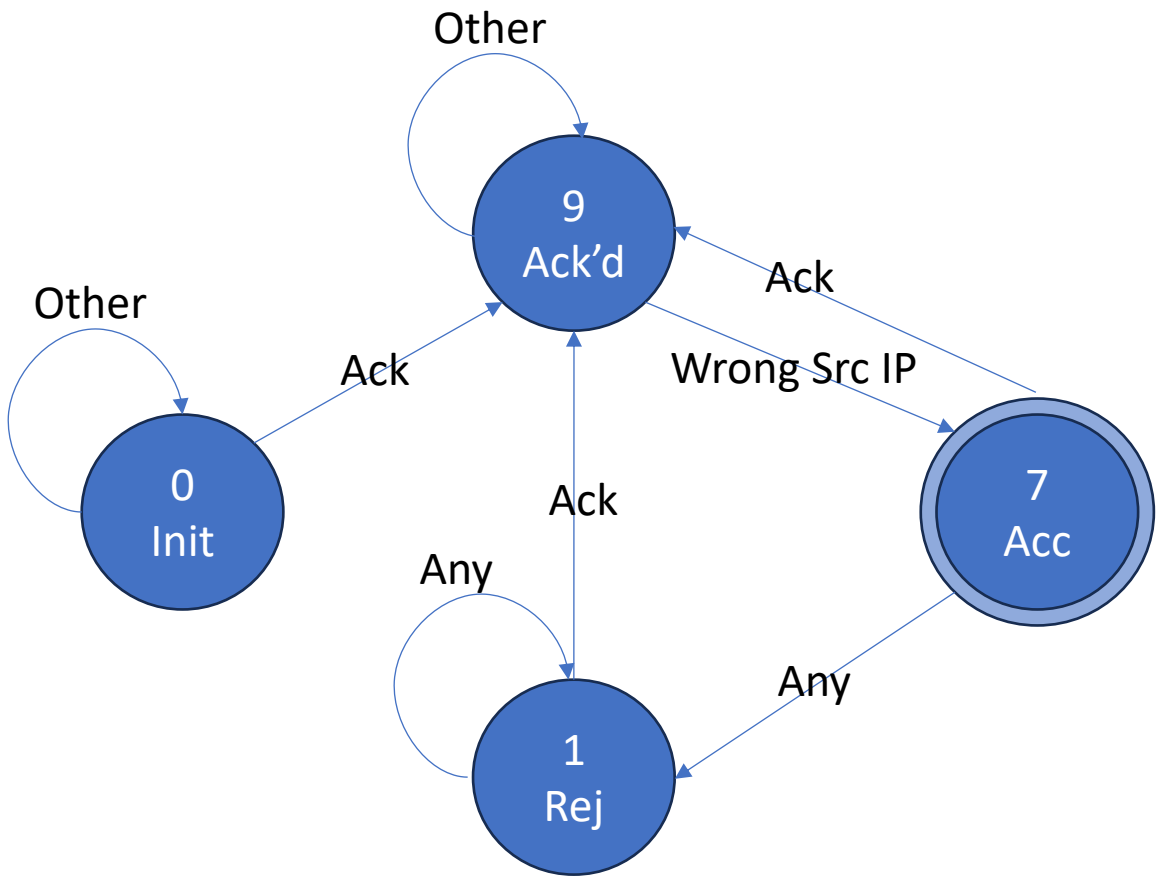
```
transition(state, input):
  if(state==0 & input==0):
    state := 1
  elif(state==1 & input==0):
    state := 2
  elif(state==2 & input==0):
    state := 3
. . .
```

Embedding State Machines in Hardware




```
Ack -> 12
Wrong Src IP -> 2
Other -> 0
```

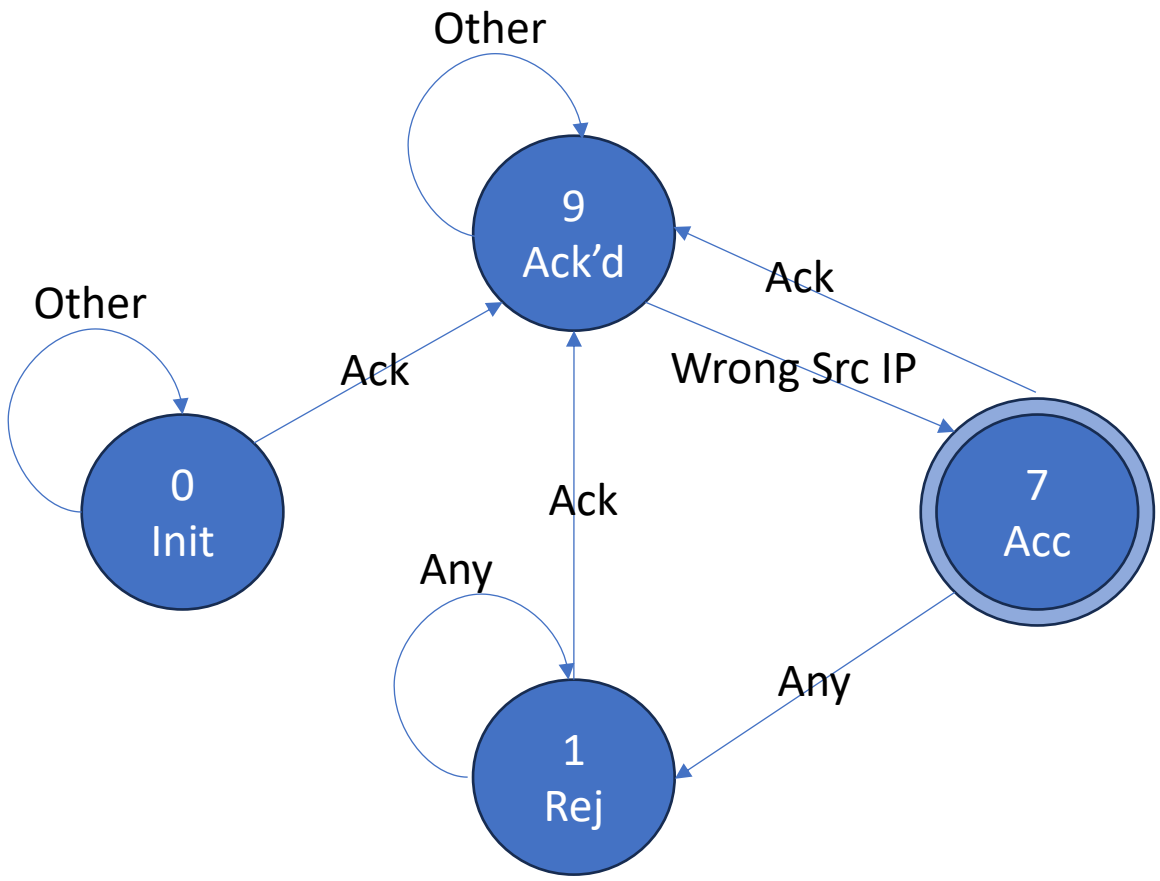
Embedding State Machines in Hardware




```
Ack -> 12
Wrong Src IP -> 2
Other -> 0
```

```
transition(state, input):
  if (state + input > 9):
    state := input xor 5
  else:
    state := state & 1
```

Embedding State Machines in Hardware




```
Ack -> 12
Wrong Src IP -> 2
Other -> 0
```

```
transition(state, input):
  if (state + input > 9):
    state := input xor 5
  else:
    state := state & 1
```

```
Example: Init, Ack -> Ack'd
00000 + 01100 > 01001
Return: 01100 xor 00101 = 01001 = 9
```

```
 Other*
.Ack(@int assigned = cip)
.Other(sip == assigned)*
.Other(sip != assigned)
```

Remove Bindings

On Ack:
assigned := cip

```
 Other*
.Ack
.Other(sip == assigned)*
.Other(sip != assigned)
```

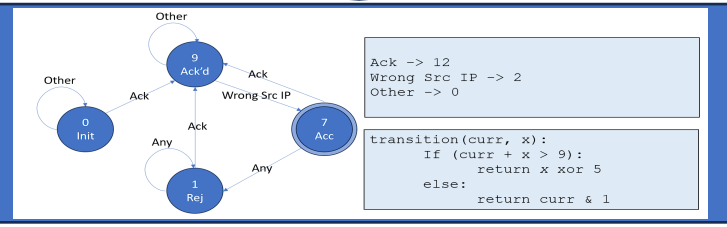
Remove Prediactes

On Ack:
assigned := cip

```
 (Other1 + Other0)*
.Ack
.Other1*
.Other0
```

Synthesize DFA

On Ack:
assigned := cip



P4 Compiler

Evaluating FLM

Evaluation Goals

- Two criterion for useful Sequence monitors were
  - Flexible
  - Line-rate
- Can FLM be used to write a wide variety of programs?
- Does the compiler work in a reasonable amount of time?

Evaluation setup

- We implemented 10 different FLM monitors
- Varying network services from simple protocols to monitoring applications
- Compile into DFA:
  4~19 States, 2~32 Symbols

1. DHCP
2. Video Fingerprinting
3. Port Knocking
4. DNS Tampering
5. DNS Tunneling
6. Distributed Storage
7. Network Coordination
8. Paxos
9. ML Training 1
10. ML Training 2

Evaluation: Ease of use (lines of code)

	FLM lines of code	P4 lines of code
1. DHCP	35	1002
2. Video Fingerprinting	41	1948
3. Port Knocking	19	795
4. DNS Tampering	27	746
5. DNS Tunneling	38	1086
6. Distributed Storage	20	925
7. Network Coordination	16	558
8. Paxos	25	1154
9. ML Training 1	21	619
10. ML Training 2	35	639

Evaluation: Ease of use (lines of code)

	FLM lines of code	P4 lines of code
1. DHCP	35	1002
2. Video Fingerprinting	41	1948
3. Port Knocking		
4. DNS Tampering		
5. DNS Tunneling		
6. Distributed Stor		
7. Network Coordi		
8. Paxos		
9. ML Training 1	21	619
10. ML Training 2	35	639

Uses only 30~40 LOC for complex patterns

Evaluation: Compilation time (seconds)

	FLM	P4
1. DHCP	7.3	11.6
2. Video Fingerprinting	36.8	18.9
3. Port Knocking	114.9	9.9
4. DNS Tampering	4.1	9.9
5. DNS Tunneling	23.2	12.5
6. Distributed Storage	7.0	10.2
7. Network Coordination	3.3	7.4
8. Paxos	556.3	10.9
9. ML Training 1	3.5	7.2
10. ML Training 2	19.5	7.4

Evaluation: Compilation time (seconds)

	FLM	P4
1. DHCP	7.3	11.6
2. Video Fingerprinting	36.8	18.9
3. Por		
4. DN		
5. DN		
6. Dis		
7. Ne		
8. Paxos		
9. ML Training 1	3.5	7.2
10. ML Training 2	19.5	7.4

Usually compiles
<2 minutes

Summary

- Presented FLM, a pattern-based language for specifying sequence monitors
- Showed a series of compilation steps to network hardware
- Proved compilation correctness for semantics of patterns
- Evaluated the system on ease of use and efficiency

Summary

- Presented FLM, a pattern-based language for specifying sequence monitors
- Showed a series of compilation steps to network hardware
- Proved compilation correctness for semantics of patterns
- Evaluated the system on ease of use and efficiency

Thank you!

Contact: aj3189@princeton.edu