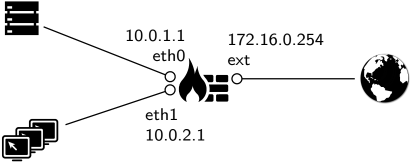
From Firewalls to Functions and Back

Lorenzo Ceragioli
Università di Pisa, Pisa, Italy
lorenzo.ceragioli@phd.unipi.it

Letterio Galletta
IMT School for Advanced Studies, Lucca, Italy
letterio.galletta@imtlucca.it

Mauro Tempesta
Università Ca' Foscari, Venezia, Italy
TU Wien, Vienna, Austria
tempesta@unive.it

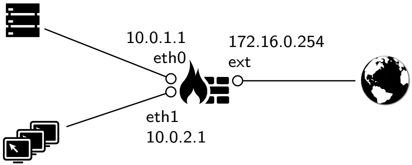
What is a Firewall?



Inspects the traffic on a node of the network, for each packet

- **accepts or drops** it
- possibly **changes the addresses** (NAT)

What is a Firewall?



Inspects the traffic on a node of the network, for each packet

- **accepts or drops** it
- possibly **changes the addresses** (NAT)

Based on a **configuration**

- **List of rules**
- Possibly using **tags**
- **Procedure**-like constructs
- **Interaction** among rules (Shadowing)

Motivations

Firewalls are a basic tool for protecting network

- **Widespread**
- **Configuration-based**
- **Different** configuration languages (`iptables`, `pf`, `ipfw`)
- It's **Hard** to configure and manage firewalls
- Cross-platform policy porting is **Harder**

Motivations

Firewalls are a basic tool for protecting network

- **Widespread**
- **Configuration-based**
- **Different** configuration languages (`iptables`, `pf`, `ipfw`)
- It's **Hard** to configure and manage firewalls
- Cross-platform policy porting is **Harder**

> Misconfigurations cause unintended behaviour
> **Possible Threats**

Transcompiltation Pipeline

Previous works:

Transcompilation Pipeline between firewall languages

- Supports `iptables`, `pf`, `ipfw` and (partially) `CISCO`-*ios*
- General approach
- Supports NAT
- Formal semantics

Transcompiltation Pipeline

Previous works:

Transcompilation Pipeline between firewall languages

- Supports `iptables`, `pf`, `ipfw` and (partially) `CISCO`-*ios*
- General approach
- Supports NAT
- Formal semantics

Why

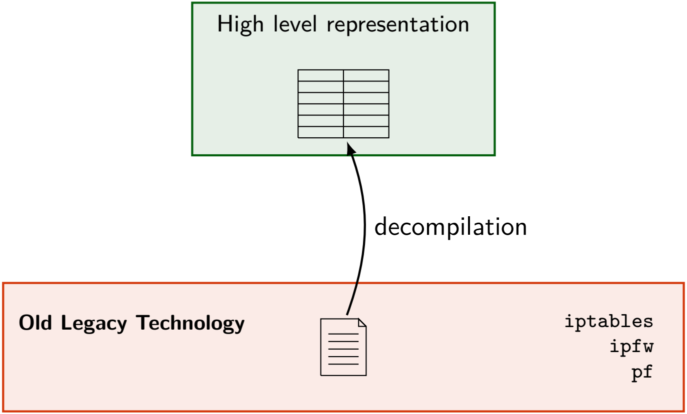
- For automated policy **porting** (first general approach!)
- For configuration **refactoring**
- Synthesis of a high level **declarative configuration**
- **Basis** for other policy management tasks

Our Goal

Old Legacy Technology

`iptables`
`ipfw`
`pf`

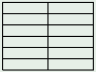
Our Goal



High level representation
decompilation
Old Legacy Technology
iptables
ipfw
pf

Our Goal

High level representation

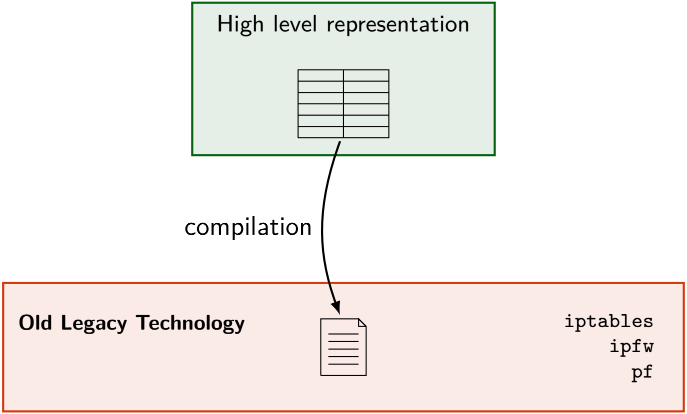


configuration update
configuration testing
configuration generation
policy verification
various representation

Old Legacy Technology

`iptables`
`ipfw`
`pf`

Our Goal


High level representation
compilation
Old Legacy Technology
iptables
ipfw
pf

IFCL — Intermediate Firewall Configuration Language

Each firewall system

- Has **its own** configuration **language**
- Makes **different evaluation steps** to process packets
- Lots of **low level** details
  - First do the NAT, than filtering or vice-versa?
  - How to express complex conditions (negated)?

IFCL — Intermediate Firewall Configuration Language

Each firewall system

- Has **its own** configuration **language**
- Makes **different evaluation steps** to process packets
- Lots of **low level** details
  - First do the NAT, than filtering or vice-versa?
  - How to express complex conditions (negated)?

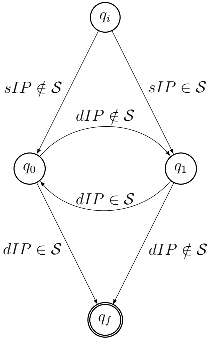
General Model

Firewall = set of rules + the evaluating procedure

IFCL — Intermediate Firewall Configuration Language

Firewall = set of rules + the evaluating procedure

Control Diagram



$\mathcal{S}$ are the addresses of the firewall

IFCL — Intermediate Firewall Configuration Language

Firewall = set of rules + the evaluating procedure

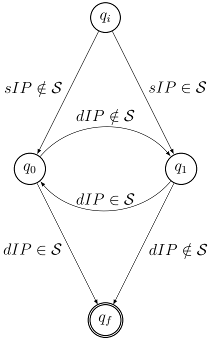
Configuration

Assigns a rulesets to each node

Ruleset : list of **rules** $r = (\phi, a)$

- $\phi(p)$: **condition**
- a : **action**
  - `ACCEPT`
  - `DROP`
  - `NAT`$(d_n,\ s_n)$
  - `MARK(m)`
  - `GOTO`(R)
  - `CALL`(R)
  - `RETURN`

Control Diagram



$\mathcal{S}$ are the addresses of the firewall

Firewall = set of rules + the evaluating procedure

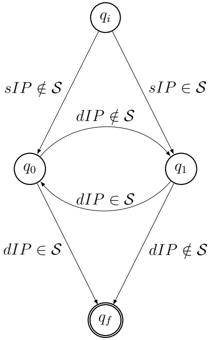
Configuration

Assigns a rulesets to each node

Ruleset : list of **rules** $r = (\phi, a)$

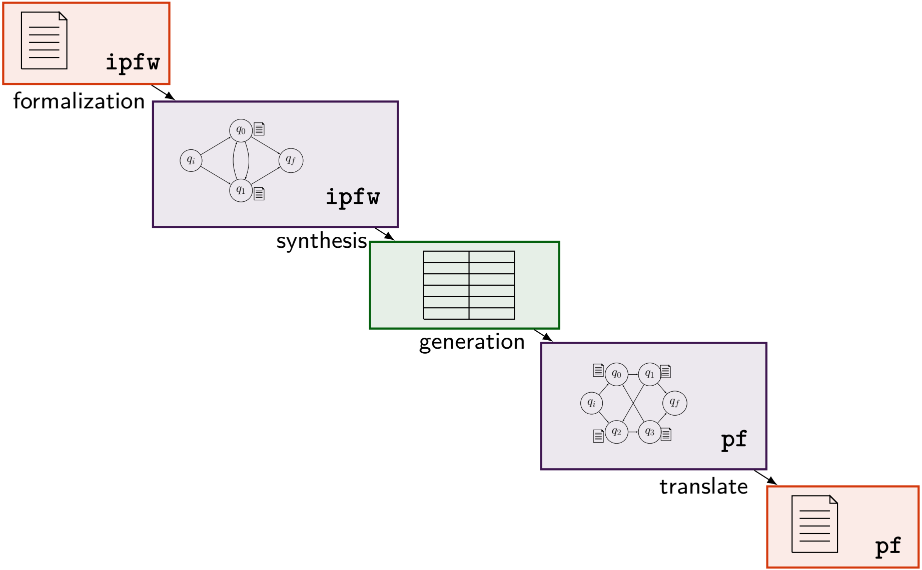
- $\phi(p)$: **condition**
- a : **action**
  - `ACCEPT`
  - `DROP`
  - `NAT`$(d_n,\ s_n)$
  - `MARK(m)`
  - ~~`GOTO`(R)~~
  - ~~`CALL`(R)~~
  - ~~`RETURN`~~

Control Diagram



$\mathcal{S}$ are the addresses of the firewall

Transcompilation Pipeline



From Firewalls to Functions and Back: The Idea

Previous implementation of the pipeline synthesis:

Compute the models of a predicate (SAT-solver)
Black-box approach (no fine tuning)

From Firewalls to Functions and Back: The Idea

Previous implementation of the pipeline synthesis:

Compute the models of a predicate (SAT-solver)
Black-box approach (no fine tuning)

Change of domain:

Function-based redefinition of the pipeline

From Firewalls to Functions and Back: The Idea

Previous implementation of the pipeline synthesis:

Compute the models of a predicate (SAT-solver)
Black-box approach (no fine tuning)

Change of domain:

Function-based redefinition of the pipeline

(Firewalls $\rightharpoonup$ Functions) :
source configuration $\mapsto$ function representing its **meaning**

(Firewalls $\leftharpoondown$ Functions) :
functional representation $\mapsto$ target configuration

From Firewalls to Functions and Back: The Idea

Previous implementation of the pipeline synthesis:

Compute the models of a predicate (SAT-solver)
Black-box approach (no fine tuning)

Change of domain:

Function-based redefinition of the pipeline

(Firewalls $\rightharpoondown$ Functions) :
source configuration $\mapsto$ function representing its **meaning**

(Firewalls $\leftharpoondown$ Functions) :
functional representation $\mapsto$ target configuration

Functions are an **handy domain**:
They allow **simple and general solutions**

Rulesets and Firewalls as Functions

$$\tau : \mathbb{P} \to \mathcal{T}(\mathbb{P}) \cup \{\bot\}$$ where

$\mathbb{P}$ network packets

$\mathcal{T}(\mathbb{P})$ transformations possibly applied to packets

$\bot$ discard of a packet

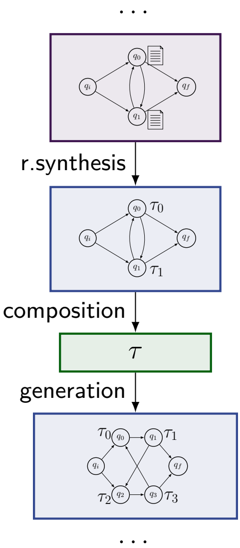
Rulesets and Firewalls as Functions

$$\tau : \mathbb{P} \rightarrow \mathcal{T}(\mathbb{P}) \cup \{\bot\} \qquad \text{where}$$

- $\mathbb{P}$ network packets
- $\mathcal{T}(\mathbb{P})$ transformations possibly applied to packets
- $\bot$ discard of a packet

New pipeline stages:

- **ruleset synthesis:** rulesets became functions
- **composition:** computes the semantics of the firewall
- **generation:** assign functions to the target nodes
- **translation:** from IFCL to pf configuration language



Rulesets and Firewalls as Functions

$$\tau : \mathbb{P} \to \mathcal{T}(\mathbb{P}) \cup \{\bot\} \qquad \text{where}$$

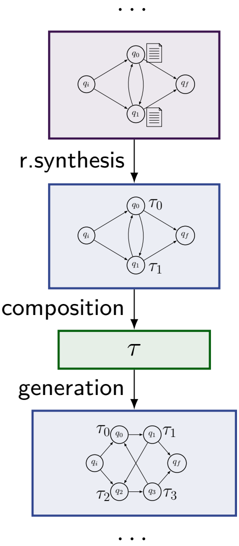
- $\mathbb{P}$ network packets
- $\mathcal{T}(\mathbb{P})$ transformations possibly applied to packets
- $\bot$ discard of a packet

New pipeline stages:

- **ruleset synthesis:** rulesets became functions
- **composition:** computes the semantics of the firewall
- **generation:** assign functions to the target nodes
- **translation:** from IFCL to `pf` configuration language

Why:

- **Parametric** w.r.t. IFCL specification
- Support **minimal control diagrams** and `MARK`
- Translation from IFCL **to target language** is trivial



Function Representation

Functions $\tau : \mathbb{P} \to \mathcal{T}(\mathbb{P}) \cup \{\bot\}$ as **sets of pairs** (P, t)

t is a transformation

P is a multi-cube of packets

Function Representation

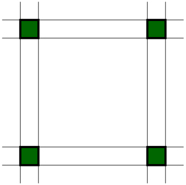
Functions $\tau : \mathbb{P} \rightarrow \mathcal{T}(\mathbb{P}) \cup \{\bot\}$ as **sets of pairs** (P, t)

t is a transformation

P is a multi-cube of packets

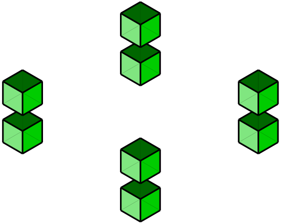
Cube :

Cartesian product of one segment for each dimension



Multi-cube :

Cartesian product of one **union of segments** for each dimension

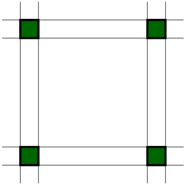


Function Representation

Functions $\tau : \mathbb{P} \rightarrow \mathcal{T}(\mathbb{P}) \cup \{\bot\}$ as **sets of pairs** (P, t)

t is a transformation

P is a multi-cube of packets

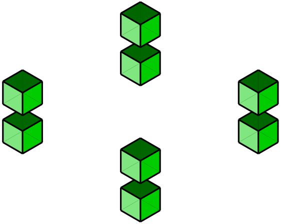


Cube :

Cartesian product of one segment for each dimension

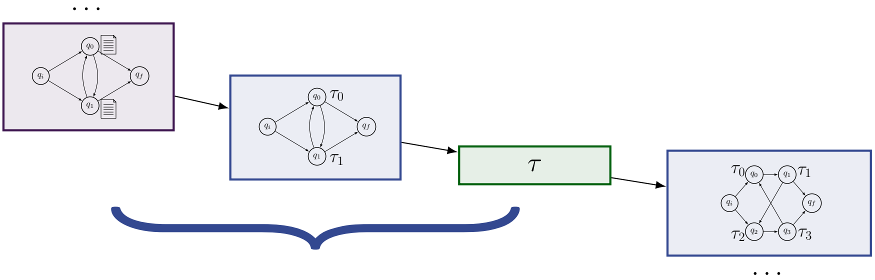
Multi-cube :

Cartesian product of one **union of segments** for each dimension



- **succinct** representation
- sets of packets verifying **rule conditions**
- sets of packets verifying **arc conditions**
- closed under **transformations**

Synthesis



Ruleset Synthesis

From a **ruleset** to a **set of pairs** (P, t)

Ruleset Synthesis

From a **ruleset** to a **set of pairs** (P, t)

We scan the ruleset rule-by-rule, keeping track of

- P packets not managed
- t transformation assigned to P

Ruleset Synthesis

From a **ruleset** to a **set of pairs** (P, t)

We scan the ruleset rule-by-rule, keeping track of

- P packets not managed
- t transformation assigned to P

Base Case: if $R = [\ ]$ just returns $\{ (P, t) \}$

Ruleset Synthesis

From a **ruleset** to a **set of pairs** (P, t)

We scan the ruleset rule-by-rule, keeping track of

P packets not managed

t transformation assigned to P

Base Case: if $R = [\,]$ just returns $\{\,(P, t)\,\}$

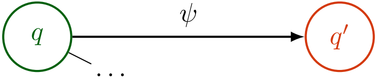
Else: take the first rule $(\phi, action)$

$$P = \begin{cases} P_s & \text{packets that verifies } \phi \\ P_n & \text{packets that do not -- managed by the \textbf{other rules}} \end{cases}$$

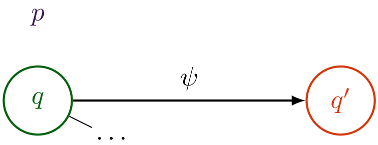
if $action$ terminates the packet processing **then** (P_s, t')

else P_s also managed by the other rules (updated transformation t')

Composition

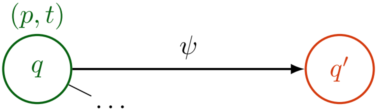


Composition



Ideally, for each $p \in \mathbb{P}$

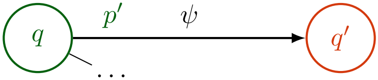
Composition



Ideally, for each $p \in \mathbb{P}$

- compute t in the first node

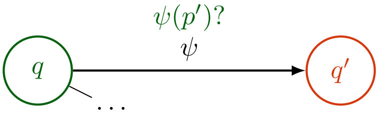
Composition



Ideally, for each $p \in \mathbb{P}$

- compute t in the first node
- compute p':
  (how p is when exits node q)

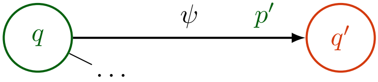
Composition



Ideally, for each $p \in \mathbb{P}$

- compute t in the first node
- compute p':
  (how p is when exits node q)
- check $\psi(p')$

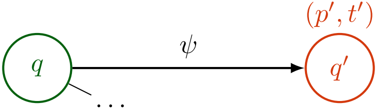
Composition



Ideally, for each $p \in \mathbb{P}$

- compute t in the first node
- compute p':
  (how p is when exits node q)
- check $\psi(p')$... if ti does then

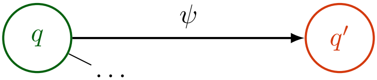
Composition



Ideally, for each $p \in \mathbb{P}$

- compute t in the first node
- compute p':
  (how p is when exits node q)
- check $\psi(p')$... if ti does then
  - compute t' in the second node

Composition

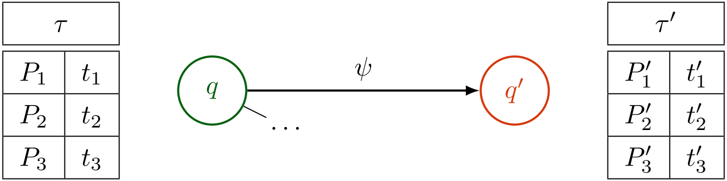


Globally $p \mapsto t$ updated with t'

Ideally, for each $p \in \mathbb{P}$

- compute t in the first node
- compute p':
  (how p is when exits node q)
- check $\psi(p')$... if ti does then
  - compute t' in the second node
  - **Overall**: $p \mapsto t$ updated by t'

Composition



Ideally, for each $p \in \mathbb{P}$

- compute t in the first node
- compute p':
  (how p is when exits node q)
- check $\psi(p')$... if ti does then
  - compute t' in the second node
  - **Overall**: $p \mapsto t$ updated by t'

Composition Algorithm:

The same,
but with Multi-cubes ...

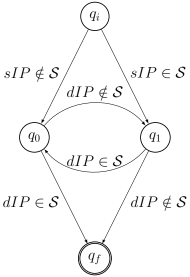
(... plus details)

Example from ipfw to pf: formalization

```
ipfw -q nat 1 config ip 151.15.185.183
ipfw -q nat 2 config redirect_port tcp 9.9.8.8:17 17
ipfw -q add 0010 nat 1 tcp from 192.168.0.0/24 to not 192.168.0.0/24
ipfw -q add 0020 nat 2 tcp from 151.15.185.183 to not 192.168.0.0/24 17
ipfw -q add 0030 allow tcp from 151.15.185.183 to not 192.168.0.0/24 out
ipfw -q add 0040 deny all from any to any
```

Example from ipfw to pf: formalization

```
ipfw -q nat 1 config ip 151.15.185.183
ipfw -q nat 2 config redirect_port tcp 9.9.8.8:17 17
ipfw -q add 0010 nat 1 tcp from 192.168.0.0/24 to not 192.168.0.0/24
ipfw -q add 0020 nat 2 tcp from 151.15.185.183 to not 192.168.0.0/24 17
ipfw -q add 0030 allow tcp from 151.15.185.183 to not 192.168.0.0/24 out
ipfw -q add 0040 deny all from any to any
```



$$
\begin{aligned}
R_0 : &(\texttt{sIP} \in 192.168.0.0/24 \wedge \texttt{dIP} \notin 192.168.0.0/24,\\
&\qquad \texttt{NAT}(\star : \star, 151.15.15.183 : \star));\\
&(\texttt{sIP} = 151.15.185.183 \wedge \texttt{dIP} \notin 192.168.0.0/24 \wedge \texttt{dPort} = 17,\\
&\qquad \texttt{NAT}(9.9.8.8 : \star, \star : \star));\\
&(true,\ \texttt{DROP})
\end{aligned}
$$

$$R_1 : \ \ldots$$

Example from ipfw to pf: ruleset synthesis

$$R_0 : (\texttt{sIP} \in 192.168.0.0/24 \wedge \texttt{dIP} \notin 192.168.0.0/24, \texttt{NAT}(\star : \star, 151.15.15.183 : \star));$$
$$(\texttt{sIP} = 151.15.185.183 \wedge \texttt{dIP} \notin 192.168.0.0/24 \wedge \texttt{dPort} = 17, \texttt{NAT}(9.9.8.8 : \star, \star : \star));$$
$$(true,\ \texttt{DROP})$$

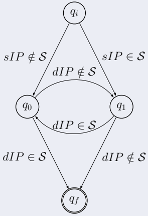
Example from `ipfw` to `pf`: ruleset synthesis

$$
\begin{aligned}
R_0 : & (\texttt{sIP} \in 192.168.0.0/24 \wedge \texttt{dIP} \notin 192.168.0.0/24, \texttt{NAT}(\star : \star, 151.15.15.183 : \star)); \\
& (\texttt{sIP} = 151.15.185.183 \wedge \texttt{dIP} \notin 192.168.0.0/24 \wedge \texttt{dPort} = 17, \texttt{NAT}(9.9.8.8 : \star, \star : \star)); \\
& (true,\ \texttt{DROP})
\end{aligned}
$$

τ_0

Received packets				**Accepted packets**			
source		**destination**		**source**		**destination**	
192.168.0.0/24	*	* \{ 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-

Example from `ipfw` to `pf`: composition



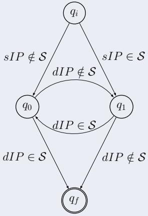
τ_0

Received packets				Accepted packets			
source		destination		source		destination	
192.168.0.0/24	*	* \{ 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-

τ_1

Received packets				Accepted packets			
source		destination		source		destination	
192.168.0.0/24	*	* \{ 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	* \{ 17 }	-	-	9.9.8.8	-

Example from `ipfw` to `pf`: composition



τ_0

Received packets				Accepted packets			
source		destination		source		destination	
192.168.0.0/24	*	* \{ 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-

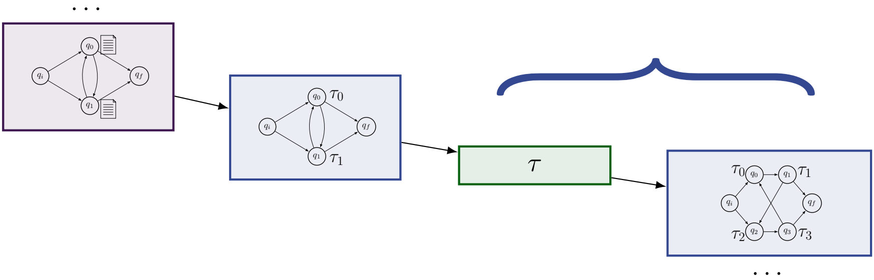
τ_1

Received packets				Accepted packets			
source		destination		source		destination	
192.168.0.0/24	*	* \{ 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	* \{ 17 }	-	-	9.9.8.8	-

↓

Received packets				Accepted packets			
source		destination		source		destination	
151.15.185.183	*	* \{ 151.15.185.183 192.168.0.0/24 }	* \{17}	-	-	-	-
192.168.0.0/24 \{192.168.0.1}	*	127.0.0.1 151.15.185.183	*	151.15.185.183	-	-	-
192.168.0.0/24 \{192.168.0.1}	*	* \{ 127.0.0.1 151.15.185.183 192.168.0.0/24 }	* \{17}	151.15.185.183	-	-	-
192.168.0.0/24 \{192.168.0.1}	*	* \{ 127.0.0.1 151.15.185.183 192.168.0.0/24 }	17	151.15.185.183	-	9.9.8.8	-
192.168.0.1	*	* \{ 127.0.0.1 151.15.185.183 192.168.0.0/24 }	*	151.15.185.183	-	-	-
151.15.185.183	*	* \{ 192.168.0.0/24 }	17	-	-	9.9.8.8	-

Generation



. . .
q_0
q_i
q_f
q_1
q_0 $\mathcal{T}_0$
q_i
q_f
q_1 $\mathcal{T}_1$
$\mathcal{T}$
$\mathcal{T}_0$ q_0
q_1 $\mathcal{T}_1$
q_i
q_f
$\mathcal{T}_2$ q_2
q_3 $\mathcal{T}_3$
. . .

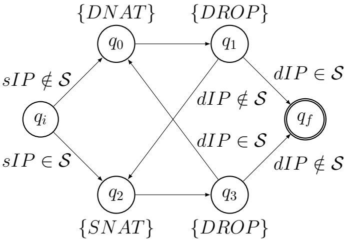
How to generate functions

Problem: not every ruleset can be assigned to each node!

- **To guarantee the final translation**
  - Simple targets: `ACCEPT`, `DROP` and `NAT`
  - Assign **Labels** to nodes:
    - `DROP`
    - `SNAT`
    - `DNAT`
- **Different expressive power**

Problem: not every ruleset can be assigned to each node!

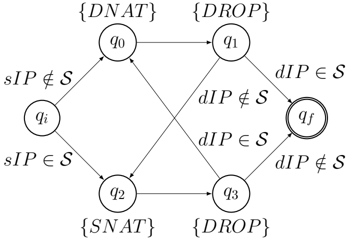
- **To guarantee the final translation**
    - Simple targets: `ACCEPT`, `DROP` and `NAT`
    - Assign **Labels** to nodes:
        - `DROP`
        - `SNAT`
        - `DNAT`
- **Different expressive power**



How to generate functions

Problem: not every ruleset can be assigned to each node!

- **To guarantee the final translation**
  - Simple targets: `ACCEPT`, `DROP` and `NAT`
  - Assign **Labels** to nodes:
    - `DROP`
    - `SNAT`
    - `DNAT`
- **Different expressive power**



Algorithm

- **For each** pair (P, t) with $t \neq \bot$
  - Find the **path**
  - **For each** node q
    - Preceding nodes $\rightarrow \mathbf{P_q}$
    - Labels in $q \rightarrow \mathbf{t_q}$
- Special management for `DROP` pairs $(P, \bot)$
  - For each node: packets **still not managed**
  - **Drop as many of these as possible**

Conclusion

The presented transcompilation approach

- Is **parametric** w.r.t. the IFCL specification
- Supports the use of **tags**
- Supports firewalls with **minimal control diagram**
- Preserves the `NAT`
- Reveals **different expressive power** of firewall languages

Conclusion

The presented transcompilation approach

- Is **parametric** w.r.t. the IFCL specification
- Supports the use of **tags**
- Supports firewalls with **minimal control diagram**
- Preserves the `NAT`
- Reveals **different expressive power** of firewall languages

Ongoing and Future Works

- **Coding** and **Testing**
- Non-trivial **multi-cube merging** procedure
- Support for **holistic network management**
- **High-level** tools for network management, **compatible with old technology**