Safely and Automatically Updating In-Network ACL Configurations with Intent Language

Bingchuan Tian
Xinyi Zhang, Ennan Zhai, Hongqiang Harry Liu, Qiaobo Ye, Chunsheng Wang, Xin Wu, Zhiming Ji, Yihong Sang, Ming Zhang, Da Yu, Chen Tian, Haitao Zheng and Ben Y. Zhao
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Security

Connectivity

deny src 3.0.0.0/8
permit src 1.0.0.0/8
...

deny src 2.0.0.0/8
...

Backbone Network

Core Service

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Security

Connectivity

Filtering Traffic Early

Backbone Network

deny src 3.0.0.0/8
permit src 1.0.0.0/8
...
deny src 2.0.0.0/8
...

Core Service

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
In-network ACL is prevalent in WAN

Note: ACLs are indeed applied to interfaces instead of a whole router. We omit interfaces for simplicity in this talk.
ACL configurations are constantly flux
ACL configurations are constantly flux

Modify existing rules

deny src 3.0.0.0/8
permit src 1.0.0.0/8
...

src: 1.0.0.3

B src: 2.0.0.5
deny src 2.0.0.0/8
...

Backbone Network

Core Service
ACL configurations are constantly flux

Modify existing rules

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

src: 1.0.0.3

B src: 2.0.0.5

deny src 2.0.0.0/8
...

Backbone Network

A

Core Service
ACL configurations are constantly flux

Modify existing rules

A
src: 1.0.0.3

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

B
src: 2.0.0.5

deny src 2.0.0.0/8
...

Backbone Network

Core Service
ACL configurations are constantly flux

Modify existing rules

A src: 1.0.0.3
B src: 2.0.0.5

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...
deny src 2.0.0.0/8
...
ACL configurations are constantly flux

Modify existing rules

A: src: 1.0.0.3

B: src: 2.0.0.5

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

deny src 2.0.0.0/8
...

Backbone Network

Core Service
ACL configurations are constantly flux

Modify existing rules

A src: 1.0.0.3

B src: 2.0.0.5

Backbone Network

Core Service
ACL configurations are constantly flux

Dig a "hole"

A
src: 1.0.0.3

B
src: 2.0.0.5

deny src 2.0.0.0/8

deny src 3.0.0.0/8
perm permit deny src 1.0.0.0/8

Backbone Network

Core Service
ACL configurations are constantly flux

Dig a "hole"

Deny src 3.0.0.0/8
Permit deny src 1.0.0.0/8
...

Permit src 2.0.0.0/16
Deny src 2.0.0.0/8
...

A src: 1.0.0.3

B src: 2.0.0.5

Backbone Network

Core Service
ACL configurations are constantly flux

Dig a "hole"

A src: 1.0.0.3

B src: 2.0.0.5

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

permit src 2.0.0.0/16
deny src 2.0.0.0/8
...

Backbone Network

Core Service
ACL configurations are constantly flux

Dig a "hole"

A src: 1.0.0.3

B src: 2.0.0.5

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

permit src 2.0.0.0/16
deny src 2.0.0.0/8
...

Backbone Network

Core Service
ACL configurations are constantly flux
ACL configurations are constantly flux

Dig a "hole"

A src: 1.0.0.3
deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

B src: 2.0.0.5

permit src 2.0.0.0/16
deny src 2.0.0.0/8
...

Backbone Network

Core Service
ACL configurations are constantly flux

Dig a "hole"

A: src: 1.0.0.3

B: src: 2.0.0.5

- deny src 3.0.0.0/8
- permit deny src 1.0.0.0/8
- ...
ACL configurations are constantly flux

Migrate ACLs

C src: 2.2.2.2

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

permit src 2.0.0.0/16
deny src 2.0.0.0/8
...

Backbone Network

Core Service
ACL configurations are constantly flux

Migrate ACLs

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

src: 2.2.2.2

permit src 2.0.0.0/16
deny src 2.0.0.0/8
...

Backbone Network

Core Service
ACL configurations are constantly flux

Migrate ACLs

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

src: 2.2.2.2

Backbone Network

permit src 2.0.0.0/16
deny src 2.0.0.0/8
...

Core Service
ACL configurations are constantly flux

Migrate ACLs

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
... 

src: 2.2.2.2

permit src 2.0.0.0/16
deny src 2.0.0.0/8
...

Backbone Network

Core Service
ACL configurations are constantly flux

Migrate ACLs

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

deny src 2.0.0.0/8
permit src 2.0.0.0/16
permit src 2.0.0.0/16
...

src: 2.2.2.2
src: 2.2.2.2

Backbone Network

Core Service
ACL configurations are constantly flux

Migrate ACLs

- deny src 3.0.0.0/8
- permit deny src 1.0.0.0/8
- ...

- permit src 2.0.0.0/16
- deny src 2.0.0.0/8
- ...

src: 2.2.2.2

Backbone Network

src: 2.2.2.2
ACL configurations are constantly flux

Migrate ACLs

C

src: 2.2.2.2

deny src 3.0.0.0/8
permit deny src 1.0.0.0/8
...

permit src 2.0.0.0/16
deny src 2.0.0.0/8
...

src: 2.2.2.2

Backbone Network

Core Service
ACL configurations are constantly flux

ACL update is error-prone

Backbone Network

Core Service
Example: ACL updates is error-prone
Example: ACL updates is error-prone
Example: ACL updates is error-prone
Example: ACL updates is error-prone

deny dst 10.0.0.0/15
permit all
Example: ACL updates is error-prone

10.0.0.0/15 = 10.0.0.0/16 + 10.1.0.0/16

deny dst 10.0.0.0/15
permit all

Public IP

A

B

C

10.0.0.0/16

10.1.0.0/16
Example: ACL updates is error-prone

\[10.0.0.0/15 = 10.0.0.0/16 + 10.1.0.0/16\]

- [Image of network diagram with nodes A, B, and C, and IP addresses 10.0.0.0/16 and 10.1.0.0/16]

  - **Public IP**
  - **deny dst 10.0.0.0/15**
  - **permit all**

- **10.0.0.0/16**
- **10.1.0.0/16**
Example: ACL updates is error-prone

deny dst 10.0.0.0/15
permit all

src: 20.0.0.1
dst: 10.0.0.1

src: 20.0.0.1
dst: 10.1.0.1

Public IP

10.0.0.0/16

10.1.0.0/16
Example: ACL updates is error-prone

deny dst 10.0.0.0/15
permit all

src: 20.0.0.1
dst: 10.1.0.1

10.0.0.0/16

10.1.0.0/16
Example: ACL updates is error-prone

deny dst 10.0.0.0/15
permit all

Public IP

src: 20.0.0.1
dst: 10.1.0.1

10.0.0.0/16

10.1.0.0/16
Example: ACL updates is error-prone

deny dst 10.0.0.0/15
permit all

Public IP

Clean up router A

10.0.0.0/16

10.1.0.0/16
Example: ACL updates is error-prone

deny dst 10.0.0.0/15
permit all

Public IP

Clean up router A
Example: ACL updates is error-prone

- **source**: 20.0.0.1
  - **destination**: 10.0.0.1

- **source**: 20.0.0.1
  - **destination**: 10.1.0.1

- **source**: 20.0.0.1
  - **destination**: 10.0.0.0/15

- **source**: 10.0.0.0/16
  - **destination**: 10.1.0.0/16

- **source**: 10.0.0.0/16
  - **destination**: 10.1.0.0/16

**Network Diagram**:
- **Router A** connected to Public IP.
- **Router B** connected to **Router A** and **Router C**.
- **Router C** connected to **Router B**.

**Actions**:
- **Deny** 10.0.0.0/15
- **Permit** all other traffic.
Example: ACL updates is error-prone

- **Public IP**
  - src: 20.0.0.1
dst: 10.1.0.1

- Clean up router A

- **B**
  - deny dst 10.0.0.0/15
  - permit all

- **C**
  - 10.1.0.0/16
  - deny dst 10.0.0.0/15
  - permit all

- **10.0.0.0/16**

- **10.1.0.0/16**
Example: ACL updates is error-prone

Public IP

Clean up router A

deny dst 10.0.0.0/15
permit all

src: 20.0.0.1
dst: 10.0.0.1

10.0.0.0/16

10.1.0.0/16

src: 20.0.0.1
dst: 10.1.0.1

deny dst 10.0.0.0/15
permit all
Example: ACL updates is error-prone
Example: ACL updates is error-prone
Example: ACL updates is error-prone
Example: ACL updates is error-prone
Example: ACL updates is error-prone

Public IP

Clean up router A

deny dst 10.0.0.0/15
permit all

src: 10.0.0.1
dst: 10.1.0.1

10.0.0.0/16

10.1.0.0/16
Example: ACL updates is error-prone
Example: ACL updates is error-prone

deny dst 10.0.0.0/15
permit all

src: 10.0.0.1
dst: 10.1.0.1

src: 10.0.0.1
dst: 10.1.0.1

Public IP

Clean up router A
Example: ACL updates is error-prone

deny dst 10.0.0.0/15
permit all

deny dst 10.0.0.0/15
permit all

src: 10.0.0.1
dst: 10.1.0.1

10.0.0.0/16

10.1.0.0/16

src: 10.0.0.1
dst: 10.1.0.1

Public IP

Clean up router A
Example: ACL updates is error-prone

Public IP

Oh dear!

deny dst 10.0.0.0/15
permit all

src: 10.0.0.1
dst: 10.1.0.1

deny dst 10.0.0.0/15
permit all

src: 10.0.0.1
dst: 10.1.0.1

10.0.0.0/16

10.1.0.0/16
Why ACL update is error-prone?

- In-network ACLs are tangled with routing policies
Why ACL update is error-prone?

• In-network ACLs are tangled with routing policies
• Service access becomes increasingly more complex
Why ACL update is error-prone?

- In-network ACLs are tangled with routing policies
- Service access becomes increasingly more complex
- WAN is always growing in both size and complexity
Why ACL update is harmful?

Failures caused by ACL misconfiguration is harmful
Why ACL update is harmful?

Failures caused by ACL misconfiguration is harmful

ACL misconfiguration undermines redundancy efforts
Why ACL update is harmful?

Failures caused by ACL misconfiguration is harmful

ACL misconfiguration undermines redundancy efforts

ACL misconfiguration is hard to be diagnosed
Root Reason

Operators can hardly handle too many complicated details in a large WAN
Separating what to update & how to update

Can we focus on **high-level intent** rather than **complicated detail** in an update?
Separating what to update & how to update

Can we focus on high-level intent rather than complicated detail in an update?

update (operator)
Separating what to update & how to update

Can we focus on high-level intent rather than complicated detail in an update?

what to update (operator)

how to
Separating what to update & how to update

Can we focus on high-level intent rather than complicated detail in an update?

what to update

(operator)

how to update
Separating *what to update & how to update*

Can we focus on **high-level intent** rather than **complicated detail** in an update?

*what to update (operator)*

*how to update (software)*
Separating *what to update* & *how to update*

Can we focus on **high-level intent** rather than **complicated detail** in an update?

**what to update**  
(operator)

**how to update**  
/software/

what to update  
[shield icon]  
/system  
[shield icon]  
all other things
Key Challenges

• How to help operators describe their intent?
Key Challenges

• How to help operators describe their intent?

Expression ability vs abstraction level
Key Challenges

• How to help operators describe their intent?

Expression ability vs abstraction level

Oh! I cannot express a special demand in this language ...
Key Challenges

• How to help operators describe their intent?

Expression ability vs abstraction level

Oh! I cannot express a special demand in this language ...

Why I have to use such a language with so many details ...
Key Challenges

• How to help operators describe their intent?
• How to design & implement ACL update primitives?
Key Challenges

• How to help operators describe their intent?
• How to design & implement ACL update primitives?

How to check the correctness of my update plan?
Key Challenges

- How to help operators describe their intent?
- How to design & implement ACL update primitives?

- How to check the correctness of my update plan?
- How to fix a wrong update?
Key Challenges

• How to help operators describe their intent?
• How to design & implement ACL update primitives?

  How to **check** the correctness of my update plan?
  How to **fix** a wrong update?
  How to **generate** an update plan?
Key Challenges

• How to help operators describe their intent?
• How to design & implement ACL update primitives?
• How to make primitives scalable to WAN?
Key Challenges

• How to help operators describe their intent?
• How to design & implement ACL update primitives?
• How to make primitives scalable to WAN?

large asymmetry topology
Key Challenges

- How to help operators describe their intent?
- How to design & implement ACL update primitives?
- How to make primitives scalable to WAN?
Key Challenges

• How to help operators describe their intent?
• How to design & implement ACL update primitives?
• How to make primitives scalable to WAN?

![Large asymmetry topology](image1.png)

![Too many rules in an ACL](image2.png)

```
[1] deny AAA
[2] permit BBB
...
[1000] permit CCC
...
[2000] deny DDD
...
[3000] permit EEE
...
[default] deny all
```

IPv4 header space: $2^{32}+32+16+16+8 = 10^{31}$
Our Solution: Jinjing

Jinjing means *Golden Eyes* in Chinese. LAI is the abbreviation of *Language for ACL Intent.*
Our Solution: Jinjing

Program written in LAI
1. scope A, B, C
2. allow B, C
3. modify A, B, C to Φ, A, A
4. check
5. fix

Jinjing means Golden Eyes in Chinese. LAI is the abbreviation of Language for ACL Intent.
Our Solution: Jinjing

Program written in LAI
1. scope A, B, C
2. allow B, C
3. modify A, B, C to \( \Phi, a, a \)
4. check
5. fix

Jinjing means Golden Eyes in Chinese. LAI is the abbreviation of Language for ACL Intent.
Our Solution: Jinjing

Jinjing means Golden Eyes in Chinese. LAI is the abbreviation of Language for ACL Intent.
Our Solution: Jinjing

Program written in LAI
1. scope A, B, C
2. allow B, C
3. modify A, B, C to Φ, A, A
4. check
5. fix

Jinjing means Golden Eyes in Chinese. LAI is the abbreviation of Language for ACL Intent.
Our Solution: Jinjing

Jinjing means *Golden Eyes* in Chinese. LAI is the abbreviation of *Language for ACL Intent.*
Our Solution: Jinjing

Jinjing means Golden Eyes in Chinese. LAI is the abbreviation of Language for ACL Intent.

Feedback
Incorrect
Fix Plan:
1. add permit src 10.0.0.0/15 to B
2. add permit src 10.0.0.0/15 to C

Program written in LAI
1. scope A, B, C
2. allow B, C
3. modify A, B, C to Φ, A, A
4. check
5. fix
Our Solution: Jinjing

Feedback
Incorrect
Fix Plan:
add permit src 10.0.0.0/15 to B
add permit src 10.0.0.0/15 to C

Jinjing means Golden Eyes in Chinese. LAI is the abbreviation of Language for ACL Intent.
Our Solution: Jinjing

Feedback
Incorrect
Fix Plan:
add permit src 10.0.0.0/15 to B
add permit src 10.0.0.0/15 to C

Jinjing means Golden Eyes in Chinese. LAI is the abbreviation of Language for ACL Intent.
Using Jinjing to fix a wrong ACL update

Feedback
Incorrect
Fix Plan:
add permit src 10.0.0.0/15 to B
add permit src 10.0.0.0/15 to C

deny dst 10.0.0.0/15
permit all
Using Jinjing to fix a wrong ACL update

Feedback
Incorrect
Fix Plan:
add permit src 10.0.0.0/15 to B
add permit src 10.0.0.0/15 to C

permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all

permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all
Using Jinjing to fix a wrong ACL update

Feedback
Incorrect
Fix Plan:
**add permit src** 10.0.0.0/15 **to** B
**add permit src** 10.0.0.0/15 **to** C

permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all

src: 10.0.0.1
dst: 10.1.0.1
Using Jinjing to fix a wrong ACL update

Feedback
Incorrect
Fix Plan:
add permit src 10.0.0.0/15 to B
add permit src 10.0.0.0/15 to C

permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all

src: 10.0.0.1
dst: 10.1.0.1

10.0.0.0/16
10.1.0.0/16
Using Jinjing to fix a wrong ACL update

Feedback
Incorrect
Fix Plan:
add permit src 10.0.0.0/15 to B
add permit src 10.0.0.0/15 to C

permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all

src: 10.0.0.1
dst: 10.1.0.1

permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all
Using Jinjing to fix a wrong ACL update

Feedback
Incorrect
Fix Plan:
add permit src 10.0.0.0/15 to B
add permit src 10.0.0.0/15 to C
permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all
permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all
src: 10.0.0.1
dst: 10.1.0.1
Using Jinjing to fix a wrong ACL update

Feedback
Incorrect
Fix Plan:
add permit src 10.0.0.0/15 to B
add permit src 10.0.0.0/15 to C

permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all

permit src 10.0.0.0/15
deny dst 10.0.0.0/15
permit all

src: 10.0.0.1
dst: 10.1.0.1

It works!
Outline

• Introduction and motivation

• LAI: Language for ACL update intent

• Three primitives: check, fix, and generate

• Experience
Intent language is necessary

- I want to check the correctness of an ACL update
- I want to fix an incorrect ACL update plan
- I want to generate an ACL update plan from scratch
Intent language is necessary

But, how can I express my intent?
Intent language is necessary

But, how can I express my intent?

Our language should be:
- Specific to common ACL update process
- High-level enough to use
Common Process: specifying update scope

Intent

scope A, B, C

Public IP

A

B

C

10.0.0.0/16

10.1.0.0/16
Common Process: specifying update scope
Common Process: specifying update scope
Common Process: specifying update scope
Common Process: specifying update scope

By any means, I cannot realize an ACL update occurs

Intent

scope A, B, C

Public IP

Whole WAN

10.0.0.0/16

10.1.0.0/16
Common Process: specifying candidate routers

Intent
scope A, B, C
allow B, C

Public IP

A

B

C

10.0.0.0/16

10.1.0.0/16

Alibaba Cloud Services Partner
Common Process: specifying candidate routers

Plan to deploy VPN

Public IP

10.0.0.0/16

10.1.0.0/16

Intent

scope A, B, C
allow B, C
Common Process: specifying candidate routers

Plan to deploy VPN

Public IP

A: No ACL
B: allowed
C: allowed

10.0.0.0/16
10.1.0.0/16

Intent
scope A, B, C
allow B, C
Common Process: describing an update plan

Intent
scope A, B, C
allow B, C
move A.allRules to B, C
Common Process: describing an update plan

Intent
scope A, B, C
allow B, C
move A.allRules to B, C

deny dst 10.0.0.0/15
permit all
Common Process: describing an update plan

- **Intent**
  - scope A, B, C
  - allow B, C
  - move A.allRules to B, C

- **Diagram**
  - Nodes A, B, C
  - Public IP
  - Routes: 10.0.0.0/16, 10.1.0.0/16

- Filters:
  - deny dst 10.0.0.0/15
  - permit all
Common Process: specifying what to do

**Intent**
scope A, B, C
allow B, C
move A.allRules to B, C
check: trafficNoChange
if not: fix

deny dst 10.0.0.0/15
permit all
Common Process: specifying what to do

**Intent**
- scope A, B, C
- allow B, C
- move A.allRules to B, C
- check: trafficNoChange
- if not: fix

```
deny dst 10.0.0.0/15
permit all
```

```
deny dst 10.0.0.0/15
permit all
```
Intent Language: LAI

\[
\begin{align*}
\text{prog} & ::= \text{region}; \text{req}; \text{cmd} \\
\text{region} & ::= \text{scope } l<n>; \text{ allow } l<n> \\
\text{req} & ::= \text{modify } l<n> \text{ to } l<n'> \\
\text{cmd} & ::= \text{check} \\
& \quad | \quad \text{fix} \\
& \quad | \quad \text{generate} \\
\text{l<n>} & ::= \text{nil} \mid n \text{ and } l \\
\text{n} & ::= \text{device} : \text{interface} \\
\text{n'} & ::= \text{device'} : \text{interface'} \\
\text{h} & ::= \text{src prefix} \mid \text{dst prefix}
\end{align*}
\]

Note: this BNF is a simplified version.
Intent Language: LAI

```
prog ::= region; req; cmd
region ::= scope l<n>; allow l<n>
req ::= modify l<n> to l<n ‘>
cmd ::= check
    | fix
    | generate
l<n> ::= nil | n and l
n ::= device : interface
n’ ::= device’ : interface’
h ::= src prefix | dst prefix
```

Intent
scope A, B, C
allow B, C
move A.allRules to B, C
check: trafficNoChange
if not: fix

Program written in LAI
1. scope A, B, C
2. allow B, C
3. modify A, B, C to Φ, A, A
4. check
5. fix

Note: this BNF is a simplified version.
Intent Language: LAI

It is hard to implement these three primitives

```
prog ::= region; req; cmd
region ::= scope l<n>; allow l<n>
req ::= modify l<n> to l<n>'
cmd ::= check
     | fix
     | generate
l<n> ::= nil | n and l
n ::= device : interface
n' ::= device' : interface'
h ::= src prefix | dst prefix
```

Program written in LAI

1. **scope** A, B, C
2. **allow** B, C
3. **modify** A, B, C to \( \phi \), A, A
4. **check**
5. **fix**

Note: this BNF is a simplified version.
Outline

• Introduction and motivation

• LAI: Language for ACL update intent

• Three primitives: check, fix, and generate

• Experience
Outline

• Introduction and motivation

• LAI: Language for ACL update intent

• Three primitives: check, fix, and generate
  - Formal verification
  - Synthesis

• Experience
Primitive: check

- Rigorously verify whether an update plan meets operators’ intent
Primitive: **check**

- Rigorously verify whether an update plan meets operators’ intent

**desired reachability**
Primitive: check

- Rigorously verify whether an update plan meets operators’ intent

desired reachability
no side effect
Primitive: check

- Rigorously verify whether an update plan meets operators' intent

- Why state of the art does not help?
  - Data plane verification
    - Not support ACL update verification
Primitive: check

- Rigorously verify whether an update plan meets operators’ intent

- Why state of the art does not help?
  - Data plane verification
    - Not support ACL update verification
  - Control plane verification
    - Not scalable to large networks
Primitive: check

- Rigorously verify whether an update plan meets operators’ intent

- Why state of the art does not help?
  - Data plane verification
    - Not support ACL update verification
  - Control plane verification
    - Not scalable to large networks
  - Control plane compression
    - Not suitable to WAN with heterogeneous topology & configuration
Basic Design: check

deny dst 10.0.0.0/15
permit all
Basic Design: check

Intent: reachability of all traffics will not change after the update

deny dst 10.0.0.0/15
permit all

Public IP

X
A
C
B
Y
Z
10.0.0.0/16
10.1.0.0/16
Basic Design: check

Intent: reachability of all traffics will not change after the update

- Forwarding Equivalence Class (FEC)

<table>
<thead>
<tr>
<th>FEC</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X -&gt; A -&gt; B -&gt; Y</td>
</tr>
<tr>
<td>2</td>
<td>X -&gt; A -&gt; C -&gt; Z</td>
</tr>
<tr>
<td>3</td>
<td>Y -&gt; B -&gt; C -&gt; Z</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Basic Design: **check**

Intent: reachability of all traffics will not change after the update

- **Forwarding Equivalence Class (FEC)**

<table>
<thead>
<tr>
<th>FEC</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X -&gt; A -&gt; B -&gt; Y</td>
</tr>
<tr>
<td>2</td>
<td>X -&gt; A -&gt; C -&gt; Z</td>
</tr>
<tr>
<td>3</td>
<td>Y -&gt; B -&gt; C -&gt; Z</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Basic Design: check

• Use FEC 3 as an example:

<table>
<thead>
<tr>
<th>FEC</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Y -&gt; B -&gt; C -&gt; Z</td>
</tr>
</tbody>
</table>
Basic Design: check

- Use FEC 3 as an example:

<table>
<thead>
<tr>
<th>FEC</th>
<th>Traffic</th>
<th>Path Model Before Updates</th>
<th>Path Model After Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Y → B → C → Z</td>
<td>B(h) ∧ C(h)</td>
<td>B'(h) ∧ C'(h)</td>
</tr>
</tbody>
</table>

Boolean function of packet header h
Basic Design: check

- Use FEC 3 as an example:

<table>
<thead>
<tr>
<th>FEC</th>
<th>Traffic</th>
<th>Path Model Before Updates</th>
<th>Path Model After Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Y -&gt; B -&gt; C -&gt; Z</td>
<td>B(h) ∨ C(h)</td>
<td>B′(h) ∨ C′(h)</td>
</tr>
</tbody>
</table>

- Verify the combined logic formula via SMT solver
Basic Design: check

- Use FEC 3 as an example:

<table>
<thead>
<tr>
<th>FEC</th>
<th>Traffic</th>
<th>Path Model Before Updates</th>
<th>Path Model After Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Y -&gt; B -&gt; C -&gt; Z</td>
<td>B(h) ∧ C(h)</td>
<td>B′(h) ∧ C′(h)</td>
</tr>
</tbody>
</table>

- Verify the combined logic formula via SMT solver

\[ F(h) = \neg ( (B(h) ∧ C(h)) \leftrightarrow (B′(h) ∧ C′(h)) ) \]

Is there an \( h \) where two groups of ACLs have different actions
Basic Design: check

• Use FEC 3 as an example:

<table>
<thead>
<tr>
<th>FEC</th>
<th>Traffic</th>
<th>Path Model Before Updates</th>
<th>Path Model After Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Y -&gt; B -&gt; C -&gt; Z</td>
<td>B(h) ∧ C(h)</td>
<td>B’(h) ∧ C’(h)</td>
</tr>
</tbody>
</table>

Boolean function of packet header h

• Verify the combined logic formula via SMT solver

\[ F(h) = \neg \left( (B(h) \land C(h)) \leftrightarrow (B'(h) \land C'(h)) \right) \land h.\text{src} \in Y \land h.\text{dst} \in Z \]

Is there an h where two groups of ACLs have different actions, and h belongs to FEC 3?
Optimization Insight

Fact: an ACL update usually involves only a small part of rules

Note: we have another optimization to speed up the cases where a lot of rules are involved.
Optimization Insight

Fact: an ACL update usually involves only a small part of rules

Theorem 4.1
Consider two ACLs $L$ and $L'$. $L$ is equivalent to $L'$ if $R(L, D_{L,L'})$ is equivalent to $R(L', D_{L,L'})$. $D_{L,L'} = (L - L \cap L') \cup (L' - L \cap L')$, and $R(L, S) = \{ k \in L: \exists k' \in S, \exists h, m_k(h) \wedge m_{k'}(h) \}$, where $m_k(h)$ means whether packet $h$ matches the $k$-th rule in $L$.

Note: we have another optimization to speed up the cases where a lot of rules are involved.
Optimization Insight

Fact: an ACL update usually involves only a small part of rules

Theorem 4.1
Consider two ACLs $L$ and $L'$. $L$ is equivalent to $L'$ if $R(L, D_{L,L})$ is equivalent to $R(L', D_{L,L'})$. $D_{L,L'} = (L - L \bar{\cap} L') \cup (L' - L \bar{\cap} L')$, and $R(L, S) = \{ k \in L : \exists k' \in S, \exists h, m_k(h) \land m_{k'}(h) \}$, where $m_k(h)$ means whether packet $h$ matches the $k$-th rule in $L$.

Except for updated rules, encoding a bit more rules is enough!

Note: we have another optimization to speed up the cases where a lot of rules are involved.
Only Checking "delta"

```
permit dst 10.0.0.0/16
deny dst 10.0.0.0/14
permit all

permit dst 10.4.0.0/16
permit dst 10.5.0.0/16
deny dst 10.4.0.0/14
permit all
```
Only Checking "delta"

The only change: adding one rule

```
permit dst 10.0.0.0/16
deny dst 10.0.0.0/14
permit all
```

```
permit dst 10.4.0.0/16
permit dst 10.5.0.0/16
deny dst 10.4.0.0/14
permit all
```

update

```
permit dst 10.0.0.0/16
permit dst 10.1.0.0/16
deny dst 10.0.0.0/14
permit all
```

```
permit dst 10.4.0.0/16
permit dst 10.5.0.0/16
deny dst 10.4.0.0/14
permit all
```
Only Checking "delta"

permit dst 10.0.0.0/16
deny dst 10.0.0.0/14
permit all

permit dst 10.4.0.0/16
permit dst 10.5.0.0/16
deny dst 10.4.0.0/14
permit all

The only change: adding one rule

update

permit dst 10.0.0.0/16
permit dst 10.1.0.0/16
deny dst 10.0.0.0/14
permit all

permit dst 10.4.0.0/16
permit dst 10.5.0.0/16
deny dst 10.4.0.0/14
permit all

We only need to encode these rules in model
Primitive: fix

- Correct an ACL update plan if it violates operators' intent
Primitive: fix

- Correct an ACL update plan if it violates operators' intent
- Why state of the art does not help?
  - Firewall repair
    - Not support distributed ACLs
  - Routing configuration repair
    - Only support repairing IGP configurations
Primitive: **fix**

- Correct an ACL update plan if it violates operators' intent
- Why state of the art does not help?
  - Firewall repair
    - Not support distributed ACLs
  - Routing configuration repair
    - Only support repairing IGP configurations
Primitive: fix

- Correct an ACL update plan if it violates operators' intent
- Why state of the art does not help?
  - Firewall repair
    - Not support distributed ACLs
  - Routing configuration repair
    - Only support repairing IGP configurations
Primitive: fix

- Correct an ACL update plan if it violates operators' intent
- Why state of the art does not help?
  - Firewall repair
    - Not support distributed ACLs
  - Routing configuration repair
    - Only support repairing IGP configurations

Too Many Counter Examples -- $10^{30}$
Primitive: fix

- Correct an ACL update plan if it violates operators' intent
- Why state of the art does not help?
  - Firewall repair
    - Not support distributed ACLs
  - Routing configuration repair
    - Only support repairing IGP configurations

Too Many Counter Examples -- $10^{30}$
Primitive: fix

- Correct an ACL update plan if it violates operators' intent
- Why state of the art does not help?
  - Firewall repair
    - Not support distributed ACLs
  - Routing configuration repair
    - Only support repairing IGP configurations

Too Many Counter Examples -- \(10^{30}\)
Primitive: fix

- Correct an ACL update plan if it violates operators' intent
- Why state of the art does not help?
  - Firewall repair
    - Not support distributed ACLs
  - Routing configuration repair
    - Only support repairing IGP configurations

\[ O(10) \text{ rounds} \]
Primitive: \textit{generate}

- Synthesize an update plan according to operators' intent
Primitive: generate

- Synthesize an update plan according to operators' intent
- Why state of the art does not help?
  - Control plane synthesis
    - Only support BGP & OSPF
    - Not support synthesize ACL
    - Not scalable
Primitive: generate

- Synthesize an update plan according to operators' intent
- Why state of the art does not help?
  - Control plane synthesis
    - Only support BGP & OSPF
    - Not support synthesize ACL
    - Not scalable

- Straight Forward Solution: Fix from Scratch
- Optimizations: Grouping ACL & Pruning Search Tree
Outline

- Introduction and motivation
- LAI: Language for ACL update intent
- Three primitives: check, fix, and generate
- Experience
Case Study

Note: This topology is different from our production network.
Case Study

MPLS

Core Backbone

Regional Backbone

Regional Backbone

Zone

Zone

Zone

Note: This topology is different from our production network.
Case Study

MPLS

Core Backbone

BGP

Regional Backbone

Zone

Regional Backbone

Zone

Note: This topology is different from our production network.
Case Study

MPLS

Core Backbone

Regional Backbone

Zone

BGP

ACL

Regional Backbone

Zone

Note: This topology is different from our production network.
Case Study

Note: This topology is different from our production network.
Case Study

MPLS

Core Backbone

BGP

ACL

Regional Backbone

Zone

BGP

ACL

Regional Backbone

Zone

Note: This topology is different from our production network.
Case Study

MPLS

Core Backbone

BGP

ACL

Regional Backbone

Zone

ACL

Regional Backbone

Zone

Note: This topology is different from our production network.
Case Study

MPLS

Core Backbone

BGP

This migration will block a lot of traffic within the regional backbone

BGP

Note: This topology is different from our production network.
Case Study

MPLS

Core Backbone

BGP

Regional Backbone

Zone

ACL

Note: This topology is different from our production network.

This migration will block a lot of traffic within the regional backbone

BAD NEWS
- Handling this migration manually may take several weeks
- Any error can cause network outage

GOOD NEWS
+ Now we have Jinjing
Case Study

This migration will block a lot of traffic within the regional backbone

BAD NEWS
- Handling this migration manually may take several weeks
- Any error can cause network outage

GOOD NEWS
+ Now we have Jinjing

With Jinjing
+ Only express intent
+ Safely and automatically generate update plan
+ 15 minutes to generate a plan

Note: This topology is different from our production network.
Jinjing contributes in every aspect

Preparation: several weeks
Jinjing contributes in every aspect

Preparation: several weeks $\xrightarrow{\text{intent language}}$ several hours
Jinjing contributes in every aspect

Preparation: several weeks $\xrightarrow{intent\ language}$ several hours

Reliability: not guaranteed
Jinjing contributes in every aspect

Preparation: several weeks $\rightarrow$ several hours

Reliability: not guaranteed $\rightarrow$ guaranteed
Jinjing contributes in every aspect

Preparation: several weeks $\xrightarrow{intent\ language}$ several hours

Reliability: not guaranteed $\xrightarrow{three\ primitives}$ guaranteed

Efficiency: $O(10)$ hours
Jinjing contributes in every aspect

Preparation: several weeks $\xrightarrow{\text{intent language}}$ several hours

Reliability: not guaranteed $\xrightarrow{\text{three primitives}}$ guaranteed

Efficiency: $O(10)$ hours $\xrightarrow{\text{optimizations}}$ $\sim$10 minutes
Conclusion

• ACL update is error-prone
Conclusion

• ACL update is error-prone
  • Tangled with numerous routing path
  • Service access requirements are complex
  • WAN is always growing
Conclusion

• ACL update is error-prone
  • Tangled with numerous routing path
  • Service access requirements are complex
  • WAN is always growing

• We designed Jinjing to safely and automatically update ACLs
  • First intent language sufficient for operators' daily use
  • Three efficient primitives with only $O(10)$ minutes running time
Conclusion

• ACL update is error-prone
  • Tangled with numerous routing path
  • Service access requirements are complex
  • WAN is always growing

• We designed Jinjing to safely and automatically update ACLs
  • First intent language sufficient for operators’ daily use
  • Three efficient primitives with only $O(10)$ minutes running time

• Real-world deployment in large-scale WAN
  • Effectively protect WAN from ACL misconfigurations
Thanks & Questions?

• ACL update is error-prone
  • Tangled with numerous routing path
  • Service access requirements are complex
  • WAN is always growing

• We designed Jinjing to safely and automatically update ACLs
  • First intent language sufficient for operators' daily use
  • Three efficient primitives with only O(10) minutes running time

• Real-world deployment in large-scale WAN
  • Effectively protect WAN from ACL misconfigurations