verifying SDN dataplane
5590: software defined networking

anduo wang, Temple University

T 17:30-20:00

Header Space Analysis — Static Checking For Networks

HSA

header space

- general and protocol agnostic
  - extend to new protocols and new types of checks (?)

statically check

- reachability properties
  - reachability failures, forwarding loops, traffic isolation and leakage

evaluation

- verify reachability between two subnets in 13 seconds

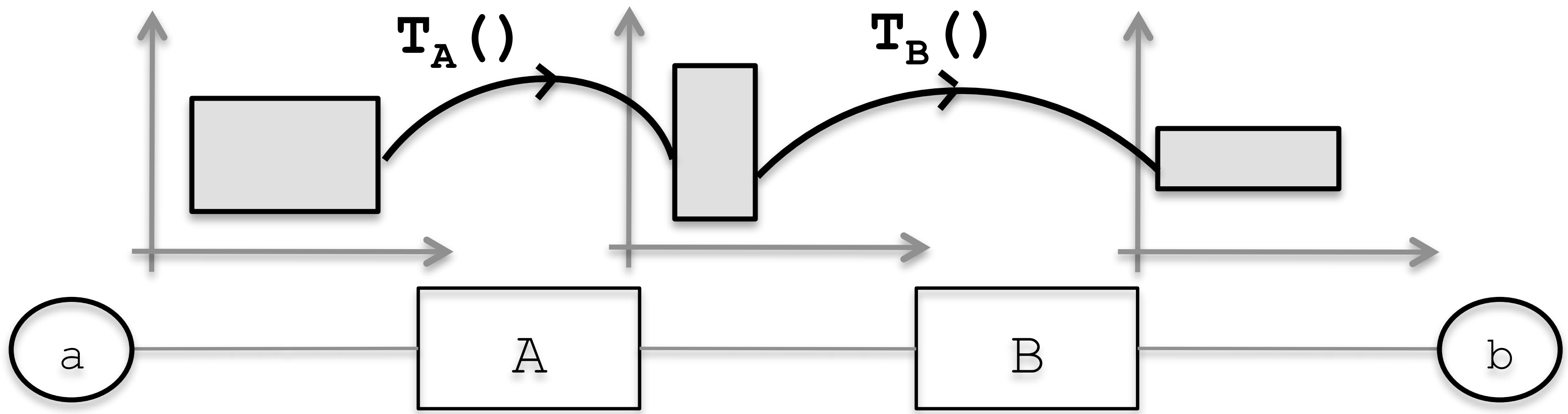
discussion (motivation)

debugging reachability is very time consuming

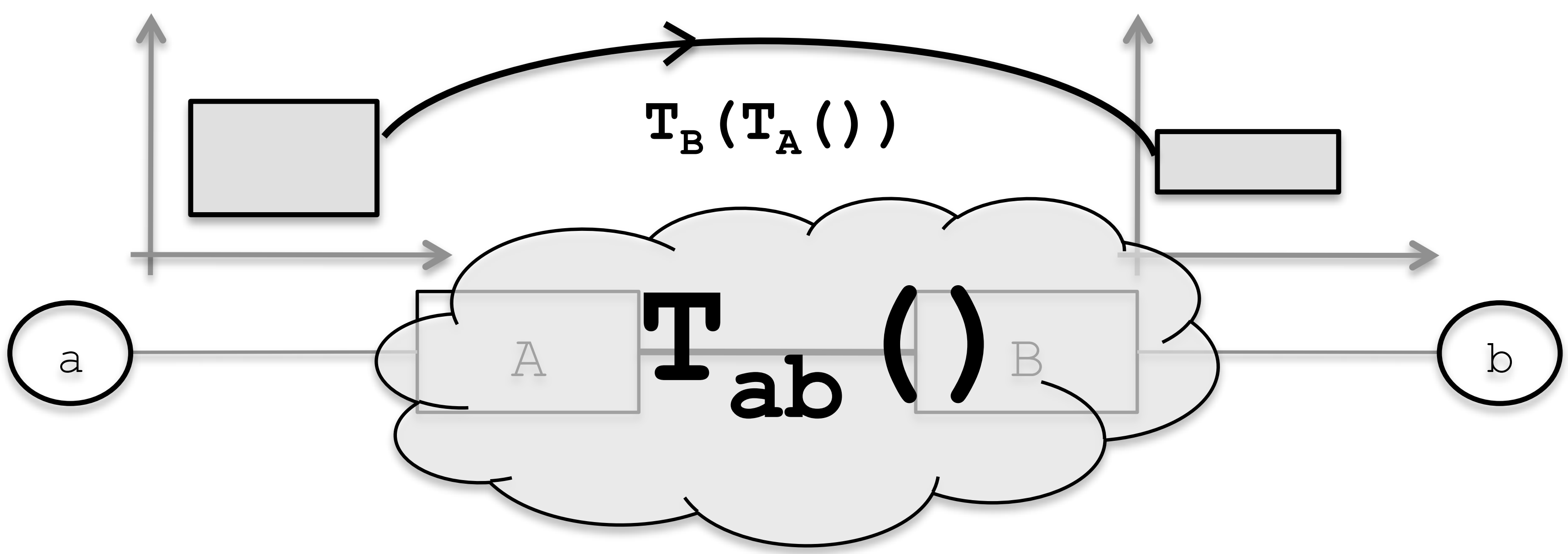
- complexity of the network state

HSA helps?

header space abstraction



(a)



header space abstraction

header space *H*

- $\{0,1\}^L$, where L is the header length
- a wildcard expression
  - sequence of L bits of 0, 1, or x (wildcard)
  - a region in header space: union of wildcard expressions

network space N

- $\{0,1\}^L \times \{1,\ldots,P\}$, where $\{1,\ldots,P\}$ is the list of ports

network transfer function

- a node transfer function $T: (h,p) \rightarrow \{(h_1, p_1), (h_2, p_2), \ldots\}$

header space abstraction

network transfer function

- a node transfer function $T: (h,p) \rightarrow \{(h_1, p_1), (h_2, p_2), \ldots\}$
- network transfer function

$$\Psi(h,p) = \begin{cases} T_1(h,p) & \text{if } p \in switch_1 \\ \ldots & \ldots \\ T_n(h,p) & \text{if } p \in switch_n \end{cases}$$

- topology transfer function

$$\Gamma(h,p) = \begin{cases} \{(h,p^*)\} & \text{if } p \text{ connected to } p^* \\ \{\} & \text{if } p \text{ is not connected.} \end{cases}$$

- multi-hop packet traversal

$$\Psi(\Gamma(\ldots(\Psi(\Gamma(h,p)\ldots)$$

using header space abstraction

an IPv4 router that forwards subnet S_1 traffic to port p_1, S_2 traffic to p_2, and S_3 traffic to p_3

$$T_r(h,p) = \begin{cases} \{(h,p_1)\} & \text{if } ip\_dst(h) \in S_1 \\ \{(h,p_2)\} & \text{if } ip\_dst(h) \in S_2 \\ \{(h,p_3)\} & \text{if } ip\_dst(h) \in S_3 \\ \{\} & \textit{otherwise.} \end{cases}$$

set operation on *H*

header space algebra

- determine how different spaces overlap
- basic set operation
  - *intersection, union, complementation, difference*

set operation on *H* — *intersection*

b_i \ b'_i	0	1	x
0	0	z	0
1	z	1	1
x	0	1	x

examples

11000xxx ∩ xx00010x = 1100010x

1100xxxx ∩ 111001xx = 11z001xx = ϕ

set operation on *H* — *union*

in general, cannot be simplified

examples

- 1100xxxx and 1000xxxx simplifies to 1x00xxxx
- 1111xxxx and 0000xxxx simplifies to ?

algorithm for logic minimization

- 10xx U 011x simplfies to $b_4\overline{b_3} \oplus \overline{b_4}b_3b_2$

set operation on H — *complementation*

$h' \leftarrow \phi$
for bit b_i in h **do**
  if $b_i \neq x$ **then**
    $h' \leftarrow h' \cup$ x...x$\overline{b_i}$x...x
  end if
end for
return h'

set operation on H — *complementation*

algorithm for computing complement for h:

$h' \leftarrow \phi$
for bit b_i in h **do**
  if $b_i \neq x$ **then**
    $h' \leftarrow h' \cup$ x...x$\overline{b_i}$x...x
  end if
end for
return h'

set operation on *H* — *complementation*

algorithm for computing complement for h:

$h' \leftarrow \phi$
for bit b_i in h **do**
  if $b_i \neq x$ **then**
    $h' \leftarrow h' \cup$ x...x$\overline{b_i}$x...x
  end if
end for
return h'

set operation on *H* — *complementation*

algorithm for computing complement for h:

$h' \leftarrow \phi$
for bit b_i in h **do**
  if $b_i \neq x$ **then**
    $h' \leftarrow h' \cup$ x...x$\overline{b_i}$x...x
  end if
end for
return h'

set operation on H — *complementation*

algorithm for computing complement for h:

$h' \leftarrow \phi$
for bit b_i in h **do**
  if $b_i \neq x$ **then**
    $h' \leftarrow h' \cup$ x...x$\overline{b_i}$x...x
  end if
end for
return h'

set operation on *H* — *complementation*

algorithm for computing complement for h:

$h' \leftarrow \phi$
for bit b_i in h **do**
  if $b_i \neq x$ **then**
    $h' \leftarrow h' \cup$ x...x$\overline{b_i}$x...x
  end if
end for
return h'

set operation on H — *complementation*

algorithm for computing complement for h:

$h' \leftarrow \phi$
for bit b_i in h **do**
  if $b_i \neq x$ **then**
    $h' \leftarrow h' \cup$ x...x$\overline{b_i}$x...x
  end if
end for
return h'

set operation on *H* — *complementation*

algorithm for computing complement for h:

$h' \leftarrow \phi$
for bit b_i in h **do**
  if $b_i \neq x$ **then**
    $h' \leftarrow h' \cup$ x...x$\overline{b_i}$x...x
  end if
end for
return h'

example

set operation on *H* — *complementation*

algorithm for computing complement for h:

$h' \leftarrow \phi$
for bit b_i in h **do**
  if $b_i \neq x$ **then**
    $h' \leftarrow h' \cup$ x...x$\overline{b_i}$x...x
  end if
end for
return h'

example

$$(100\text{xxxxx})' = 0\text{xxxxxxx} \cup \text{x}1\text{xxxxxx} \cup \text{xx}1\text{xxxxx}$$

set operation on *H* — *difference*

$A - B = A \cap B'$. For example:

$$
\begin{aligned}
&100\text{xxxxx} - 10011\text{xxx} = \\
&100\text{xxxxx} \cap (0\text{xxxxxxx} \cup \text{x}1\text{xxxxxx} \cup \text{xx}1\text{xxxxx} \\
&\cup \text{xxx}0\text{xxxx} \cup \text{xxxx}0\text{xxx}) \\
&= \phi \cup \phi \cup \phi \cup 1000\text{xxxx} \cup 100\text{x}0\text{xxx} \\
&= 1000\text{xxxx} \cup 100\text{x}0\text{xxx}.
\end{aligned}
$$

header space analysis — reachability

can packets from host a reach host b

$$R_{a \to b} = \bigcup_{a \to b \text{ paths}} \{T_n(\Gamma(T_{n-1}(......(\Gamma(T_1(h,p)...))\}$$

header space analysis — reachability

can packets from host a reach host b

$$R_{a \to b} = \bigcup_{a \to b \text{ paths}} \{T_n(\Gamma(T_{n-1}(......(\Gamma(T_1(h,p)...))\}$$

range reverse

If header $h \subset \mathcal{H}$ reached b along the $a \to S_1 \to ... \to S_{n-1} \to S_n \to b$ path, then the original header sent by a is:

$$h_a = T_1^{-1}(\Gamma(...(T_{n-1}^{-1}(\Gamma(T_n^{-1}((h,b))...)),$$

using the fact that $\Gamma = \Gamma^{-1}$.

header space analysis — reachability

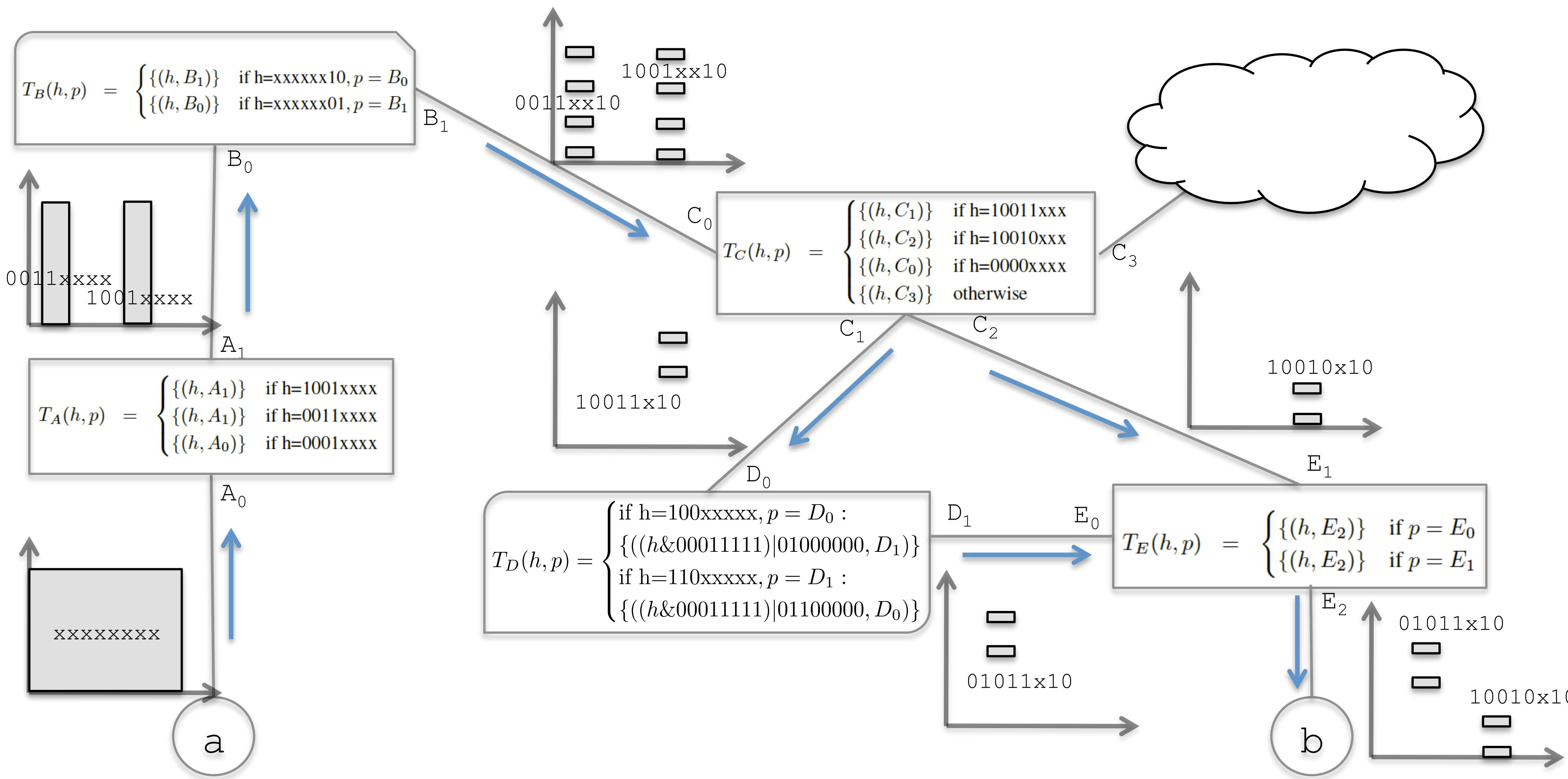


Figure 2: Example for computing reachability function from a to b. For simplicity, we assume a header length of 8 and show the first 4 bits on the x-axis and the last 4 bits on the y-axis. We show the range (output) of each transfer function composition along the paths that connect a to b. At the end, the packet headers that b will see from a are 01011x10 ∪ 10010x10.

header space analysis — reachability

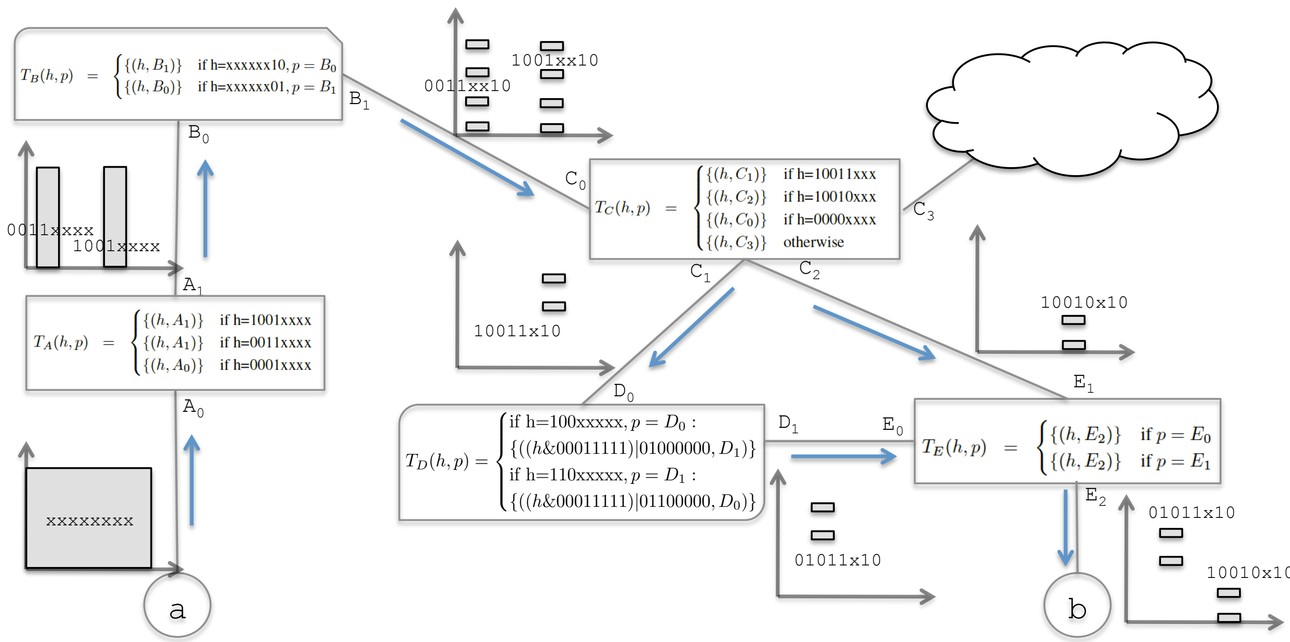


Figure 2: Example for computing reachability function from a to b. For simplicity, we assume a header length of 8 and show the first 4 bits on the x-axis and the last 4 bits on the y-axis. We show the range (output) of each transfer function composition along the paths that connect a to b. At the end, the packet headers that b will see from a are 01011x10 ∪ 10010x10.

$$R_{a\to b}(h,p) = \begin{cases} \text{if h=10010x10}, p = A_0 : \\ \{(h, E_2)\} \\ \text{if h=10011x10}, p = A_0 : \\ \{((h\&00011111)|01000000, E_2)\} \end{cases}$$

complexity — reachability

worst case complexity

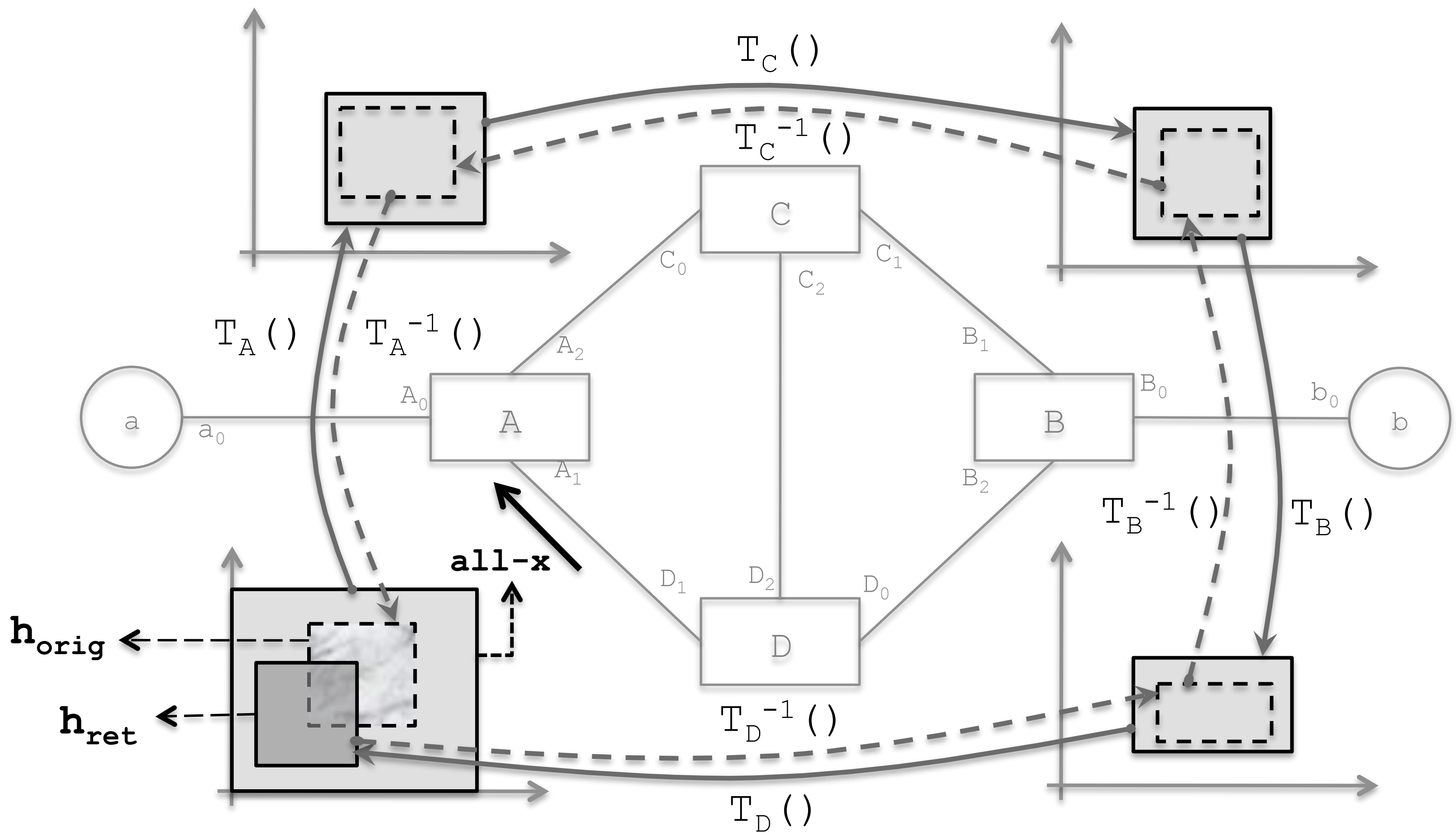
- assume input of
  - R_1 wildcard expressions, R_2 transfer function rules
- output $O(R_1 R_2)$ wildcard expressions

linear fragmentation assumption

- as packet propagates to the core of the network, the match pattern will be less specific
- cR rather than R^2 where $c \ll R$
- running time becomes $O(dR^2)$
  - d is the network diameter
  - R is the maximum number of forwarding rules in a router

brute force: $O(2^L)$

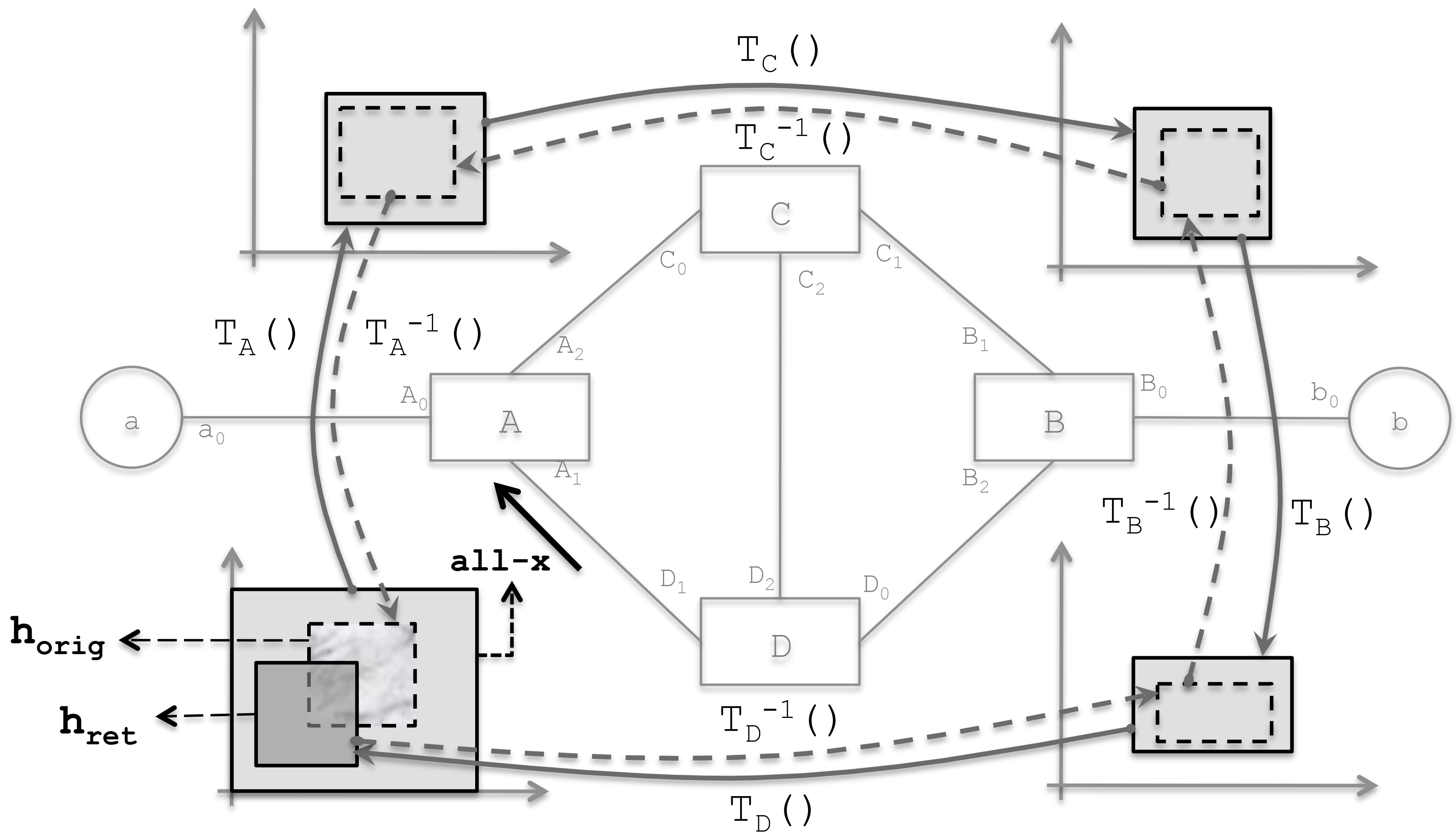
header space analysis — loop



catch loop

- inject an all-x test packet header from each port and track the packet

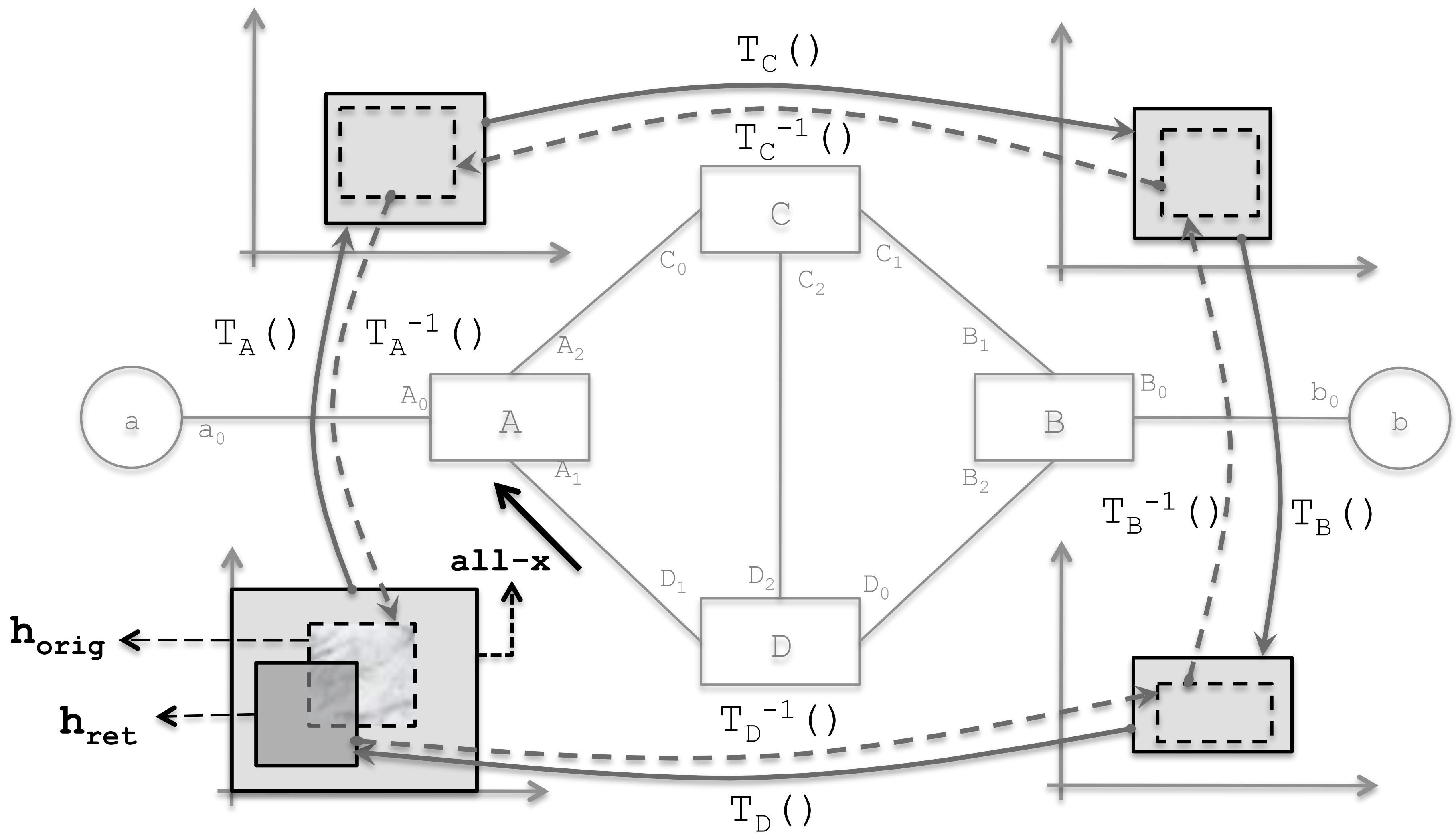
header space analysis — loop



finite loop

- $h_{ret} \cap h_{orig} = \varnothing$

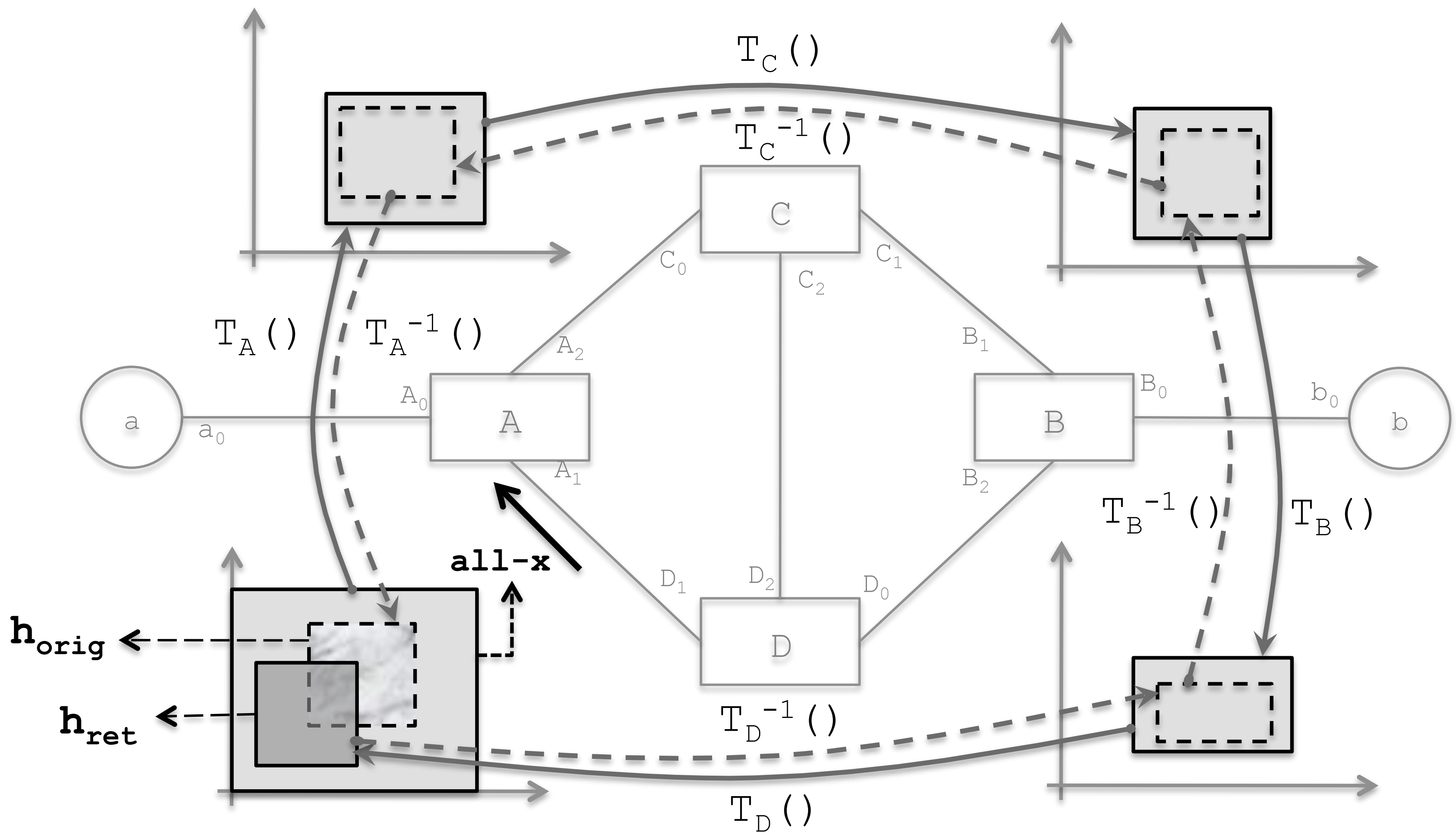
header space analysis — loop



infinite loop

- $h_{ret} \subseteq h_{orig}$

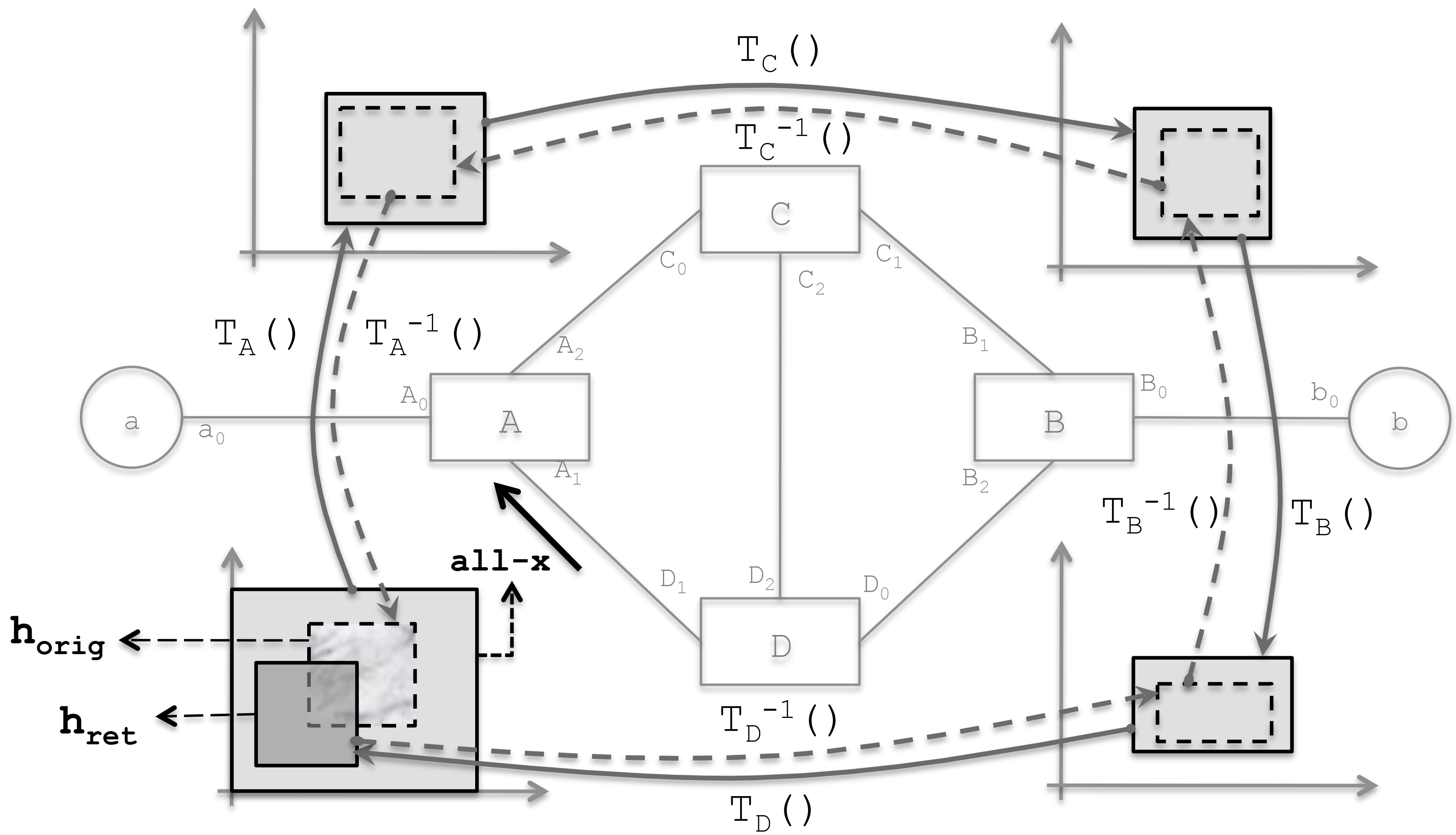
header space analysis — loop



mixed (finite and infinite)

- neither $(h_{ret} \subseteq h_{orig})$ or $h_{ret} \cap h_{orig} = \varnothing$
- $h_{ret} - h_{orig}$ is finite loop
- examine $h_{ret} \cap h_{orig}$

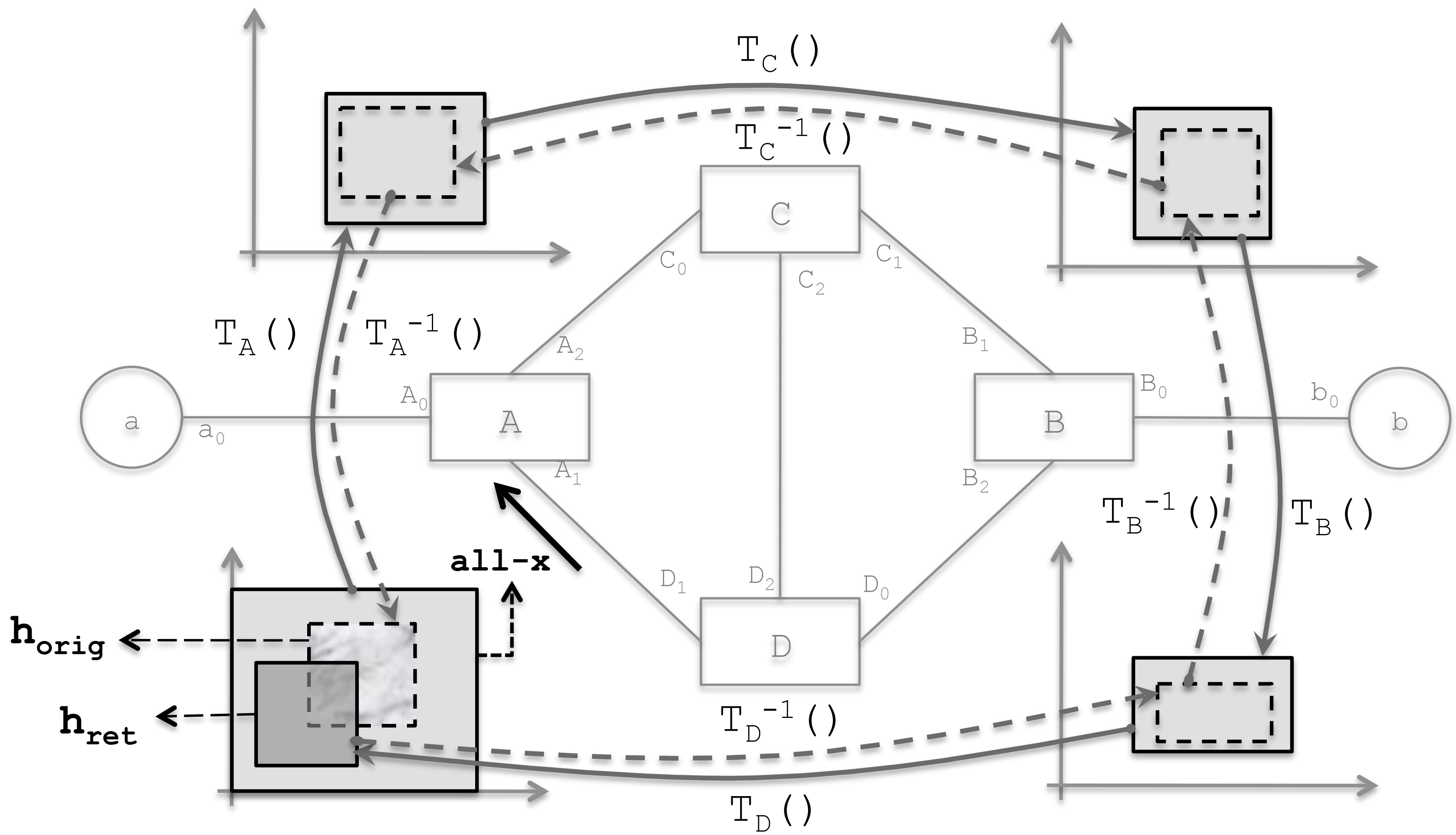
header space analysis — loop



mixed (finite and infinite)

- neither $(h_{ret} \subseteq h_{orig})$ or $h_{ret} \cap h_{orig} = \varnothing$
- h_{ret} - h_{orig} is finite loop
- examine $h_{ret} \cap h_{orig}$, what next?

header space analysis — loop



examine $h_{ret} \cap h_{orig}$

- redefine $h_{ret} := h_{ret} \cap h_{orig}$, repeat until either finite or infinite
- at most 2^L iterations

header space analysis — slice isolation

two slices, a and b with regions N_a, N_b

$$N_a = \{(\alpha_i, p_i)]_{p_i \in \mathcal{S}}\} \quad , \quad N_b = \{(\beta_i, p_i)]_{p_i \in \mathcal{S}}\}$$

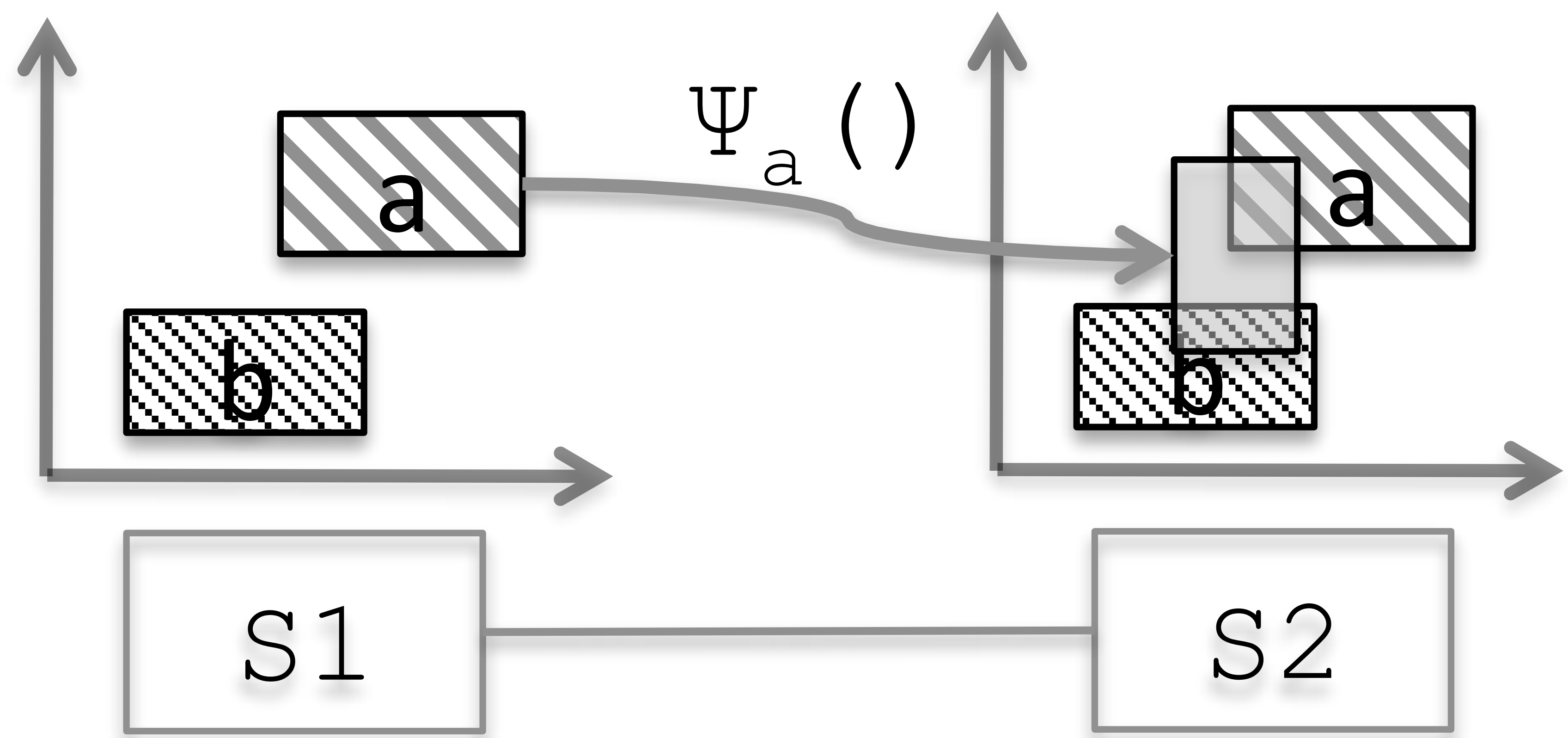
isolated $\alpha_i \cap \beta_i = \phi$

intersection

$$N_a \cap N_b = \{(\alpha_i \cap \beta_i, p_i)]_{p_i \in N_a \& p_i \in N_b}\}$$

header space analysis — slice isolation

detecting leakage



implementation

HSA is really just

- simulation + header space optimization

implementation

HSA is really just

- simulation + header space optimization

optimization

Disabled Optimization	T.F. Generation	Reach. Test	Loop Test
None	160s	12s	11s
(1) IP Table Compression	10.5x	15x	19x
(2) Lazy Subtraction	1x	>400x	>400x
(3) Dead Object Deletion	1x	8x	11x
(4) Lookup Based Search	0.9x	2x	2x
(5) Lazy T.F. evaluation	1x	1.2x	1.2x

Table 1: Impact of optimization techniques on the runtime of the reachability and loop detection algorithms.

optimization

Disabled Optimization	T.F. Generation	Reach. Test	Loop Test
None	160s	12s	11s
(1) IP Table Compression	10.5x	15x	19x
(2) Lazy Subtraction	1x	>400x	>400x
(3) Dead Object Deletion	1x	8x	11x
(4) Lookup Based Search	0.9x	2x	2x
(5) Lazy T.F. evaluation	1x	1.2x	1.2x

Table 1: Impact of optimization techniques on the runtime of the reachability and loop detection algorithms.

IP table compression

- use well known technique to reduce the number of transfer functions

optimization

Disabled Optimization	T.F. Generation	Reach. Test	Loop Test
None	160s	12s	11s
(1) IP Table Compression	10.5x	15x	19x
(2) Lazy Subtraction	1x	>400x	>400x
(3) Dead Object Deletion	1x	8x	11x
(4) Lookup Based Search	0.9x	2x	2x
(5) Lazy T.F. evaluation	1x	1.2x	1.2x

Table 1: Impact of optimization techniques on the runtime of the reachability and loop detection algorithms.

lazy subtraction

- e.g., longest prefix match with 10.1.1.x and 10.1.x.x
- allow $U\{w_i\} - U\{w_j\}$, delay the expansion of terms during intermediate steps

optimization

Disabled Optimization	T.F. Generation	Reach. Test	Loop Test
None	160s	12s	11s
(1) IP Table Compression	10.5x	15x	19x
(2) Lazy Subtraction	1x	>400x	>400x
(3) Dead Object Deletion	1x	8x	11x
(4) Lookup Based Search	0.9x	2x	2x
(5) Lazy T.F. evaluation	1x	1.2x	1.2x

Table 1: Impact of optimization techniques on the runtime of the reachability and loop detection algorithms.

dead object deletion

- lazy subtraction masks empty header space
- quick test for detecting empty header space

optimization

Disabled Optimization	T.F. Generation	Reach. Test	Loop Test
None	160s	12s	11s
(1) IP Table Compression	10.5x	15x	19x
(2) Lazy Subtraction	1x	>400x	>400x
(3) Dead Object Deletion	1x	8x	11x
(4) Lookup Based Search	0.9x	2x	2x
(5) Lazy T.F. evaluation	1x	1.2x	1.2x

Table 1: Impact of optimization techniques on the runtime of the reachability and loop detection algorithms.

lookup based search

- avoid linear search via a lookup table

optimization

Disabled Optimization	T.F. Generation	Reach. Test	Loop Test
None	160s	12s	11s
(1) IP Table Compression	10.5x	15x	19x
(2) Lazy Subtraction	1x	>400x	>400x
(3) Dead Object Deletion	1x	8x	11x
(4) Lookup Based Search	0.9x	2x	2x
(5) Lazy T.F. evaluation	1x	1.2x	1.2x

Table 1: Impact of optimization techniques on the runtime of the reachability and loop detection algorithms.

lazy evaluation of transfer function rules

- cross-product of orthogonal rules
- use commutativity to delay computation of transfer functions of one set of rules until the end

optimization

Disabled Optimization	T.F. Generation	Reach. Test	Loop Test
None	160s	12s	11s
(1) IP Table Compression	10.5x	15x	19x
(2) Lazy Subtraction	1x	>400x	>400x
(3) Dead Object Deletion	1x	8x	11x
(4) Lookup Based Search	0.9x	2x	2x
(5) Lazy T.F. evaluation	1x	1.2x	1.2x

Table 1: Impact of optimization techniques on the runtime of the reachability and loop detection algorithms.

question

- what will be the combined effects of the optimization techniques, and explain why.

evaluation

evaluation

verification of an enterprise network

- goals: usability & performance
- scenarios: reachability, loop

evaluation

verification of an enterprise network

- goals: usability & performance
- scenarios: reachability, loop

checking slice isolation

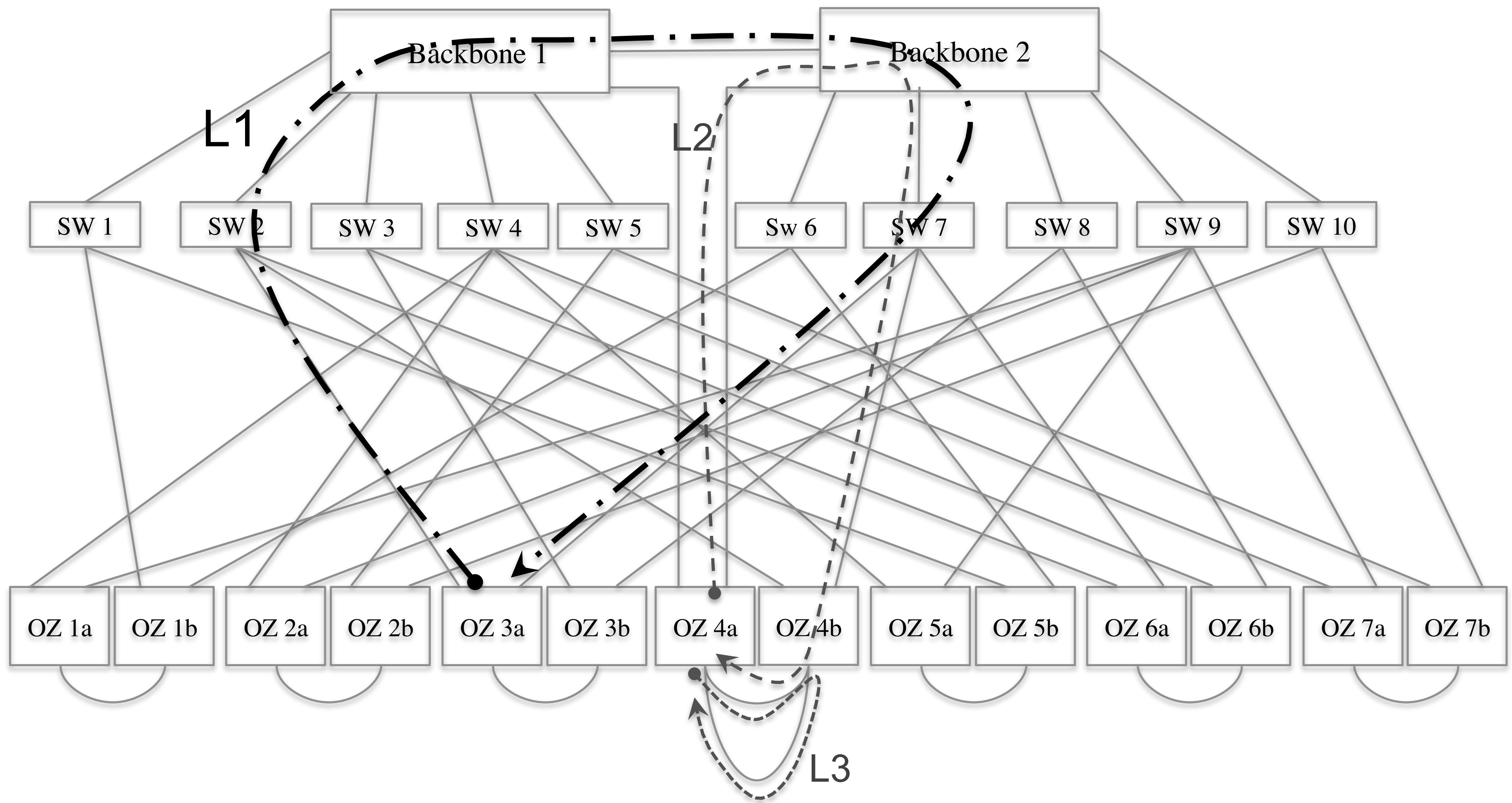
evaluation

verification of an enterprise network

- goals: usability & performance
- scenarios: reachability, loop

checking slice isolation

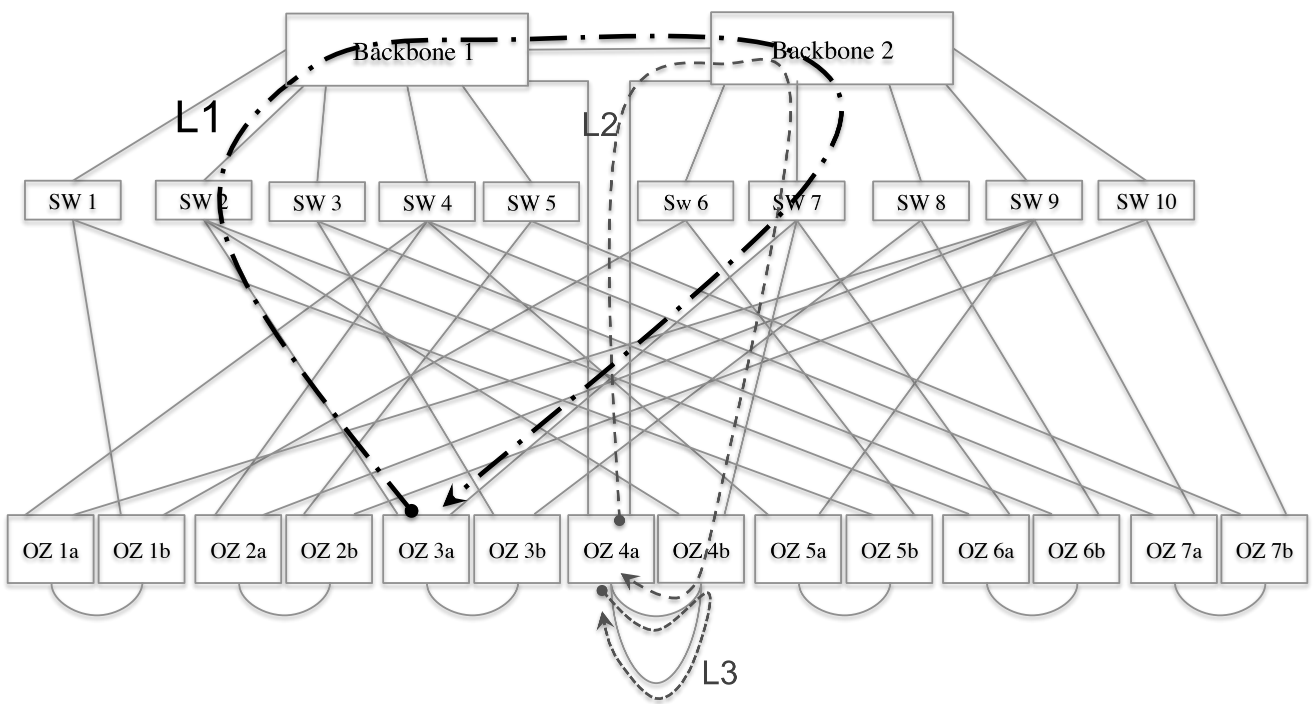
evaluation — enterprise network



setup

- Stanford backbone network
  - 757,000 forwarding entries, 1,500 ACL rules
- tests on a Macbook Pro, …

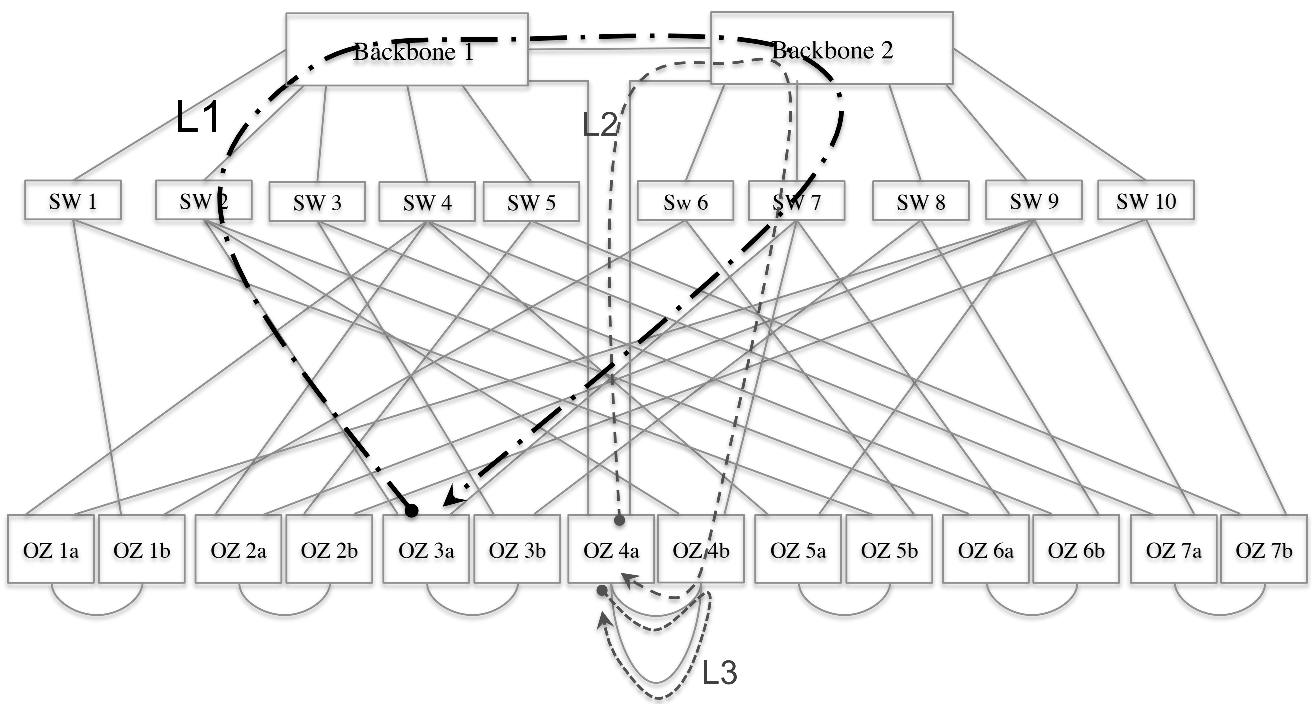
evaluation — loops



- injected test packets from 30 ports
- found 12 infinite loop paths

Time to generate Network and Topology Transfer Function	151 s
Runtime of loop detection test (30 ports)	560 s
Average per port runtime	18.6 s
Max per port runtime	135 s
Min per port runtime	8 s
Average runtime of reachability test	13 s

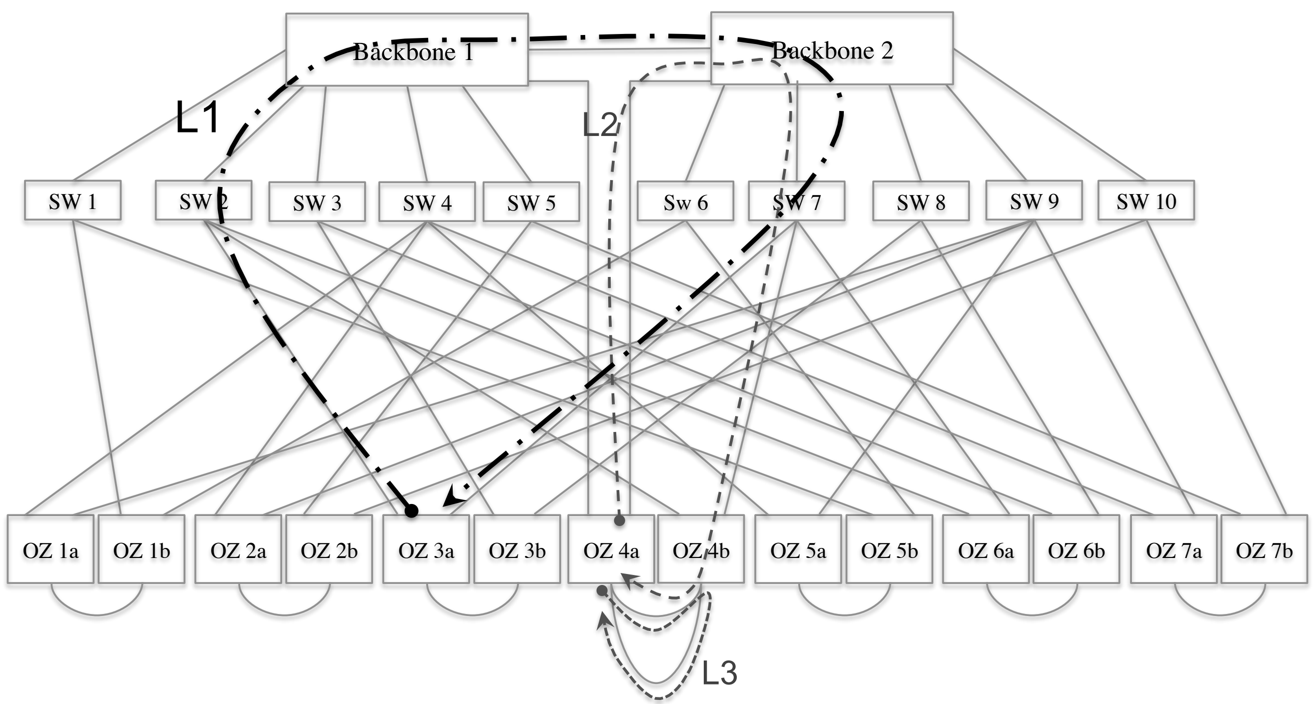
evaluation — loops



Time to generate Network and Topology Transfer Function	151 s
Runtime of loop detection test (30 ports)	560 s
Average per port runtime	18.6 s
Max per port runtime	135 s
Min per port runtime	8 s
Average runtime of reachability test	13 s

- injected test packets from 30 ports
- found 12 infinite loop paths
- weakness?

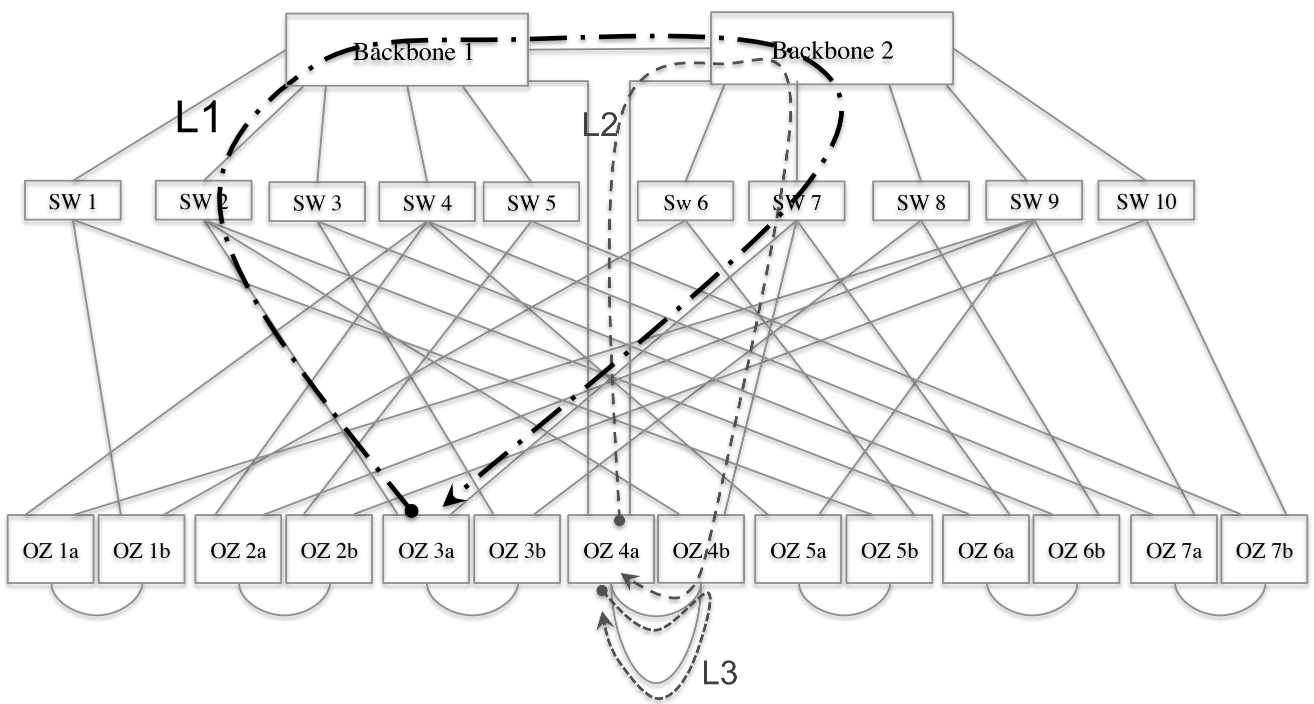
evaluation — reachability

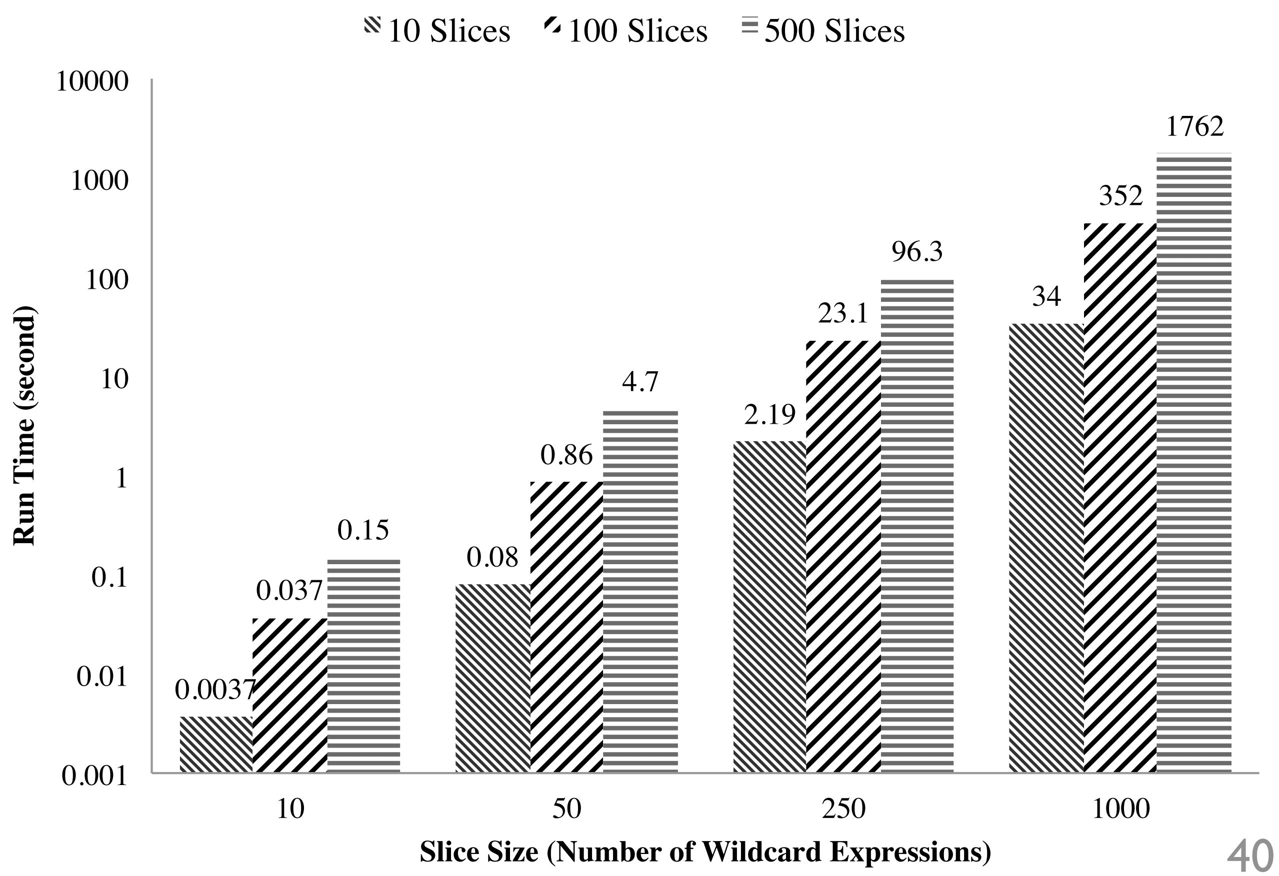


Time to generate Network and Topology Transfer Function	151 s
Runtime of loop detection test (30 ports)	560 s
Average per port runtime	18.6 s
Max per port runtime	135 s
Min per port runtime	8 s
Average runtime of reachability test	13 s

- verify intended security restrictions
  - commented by the admin in the config file

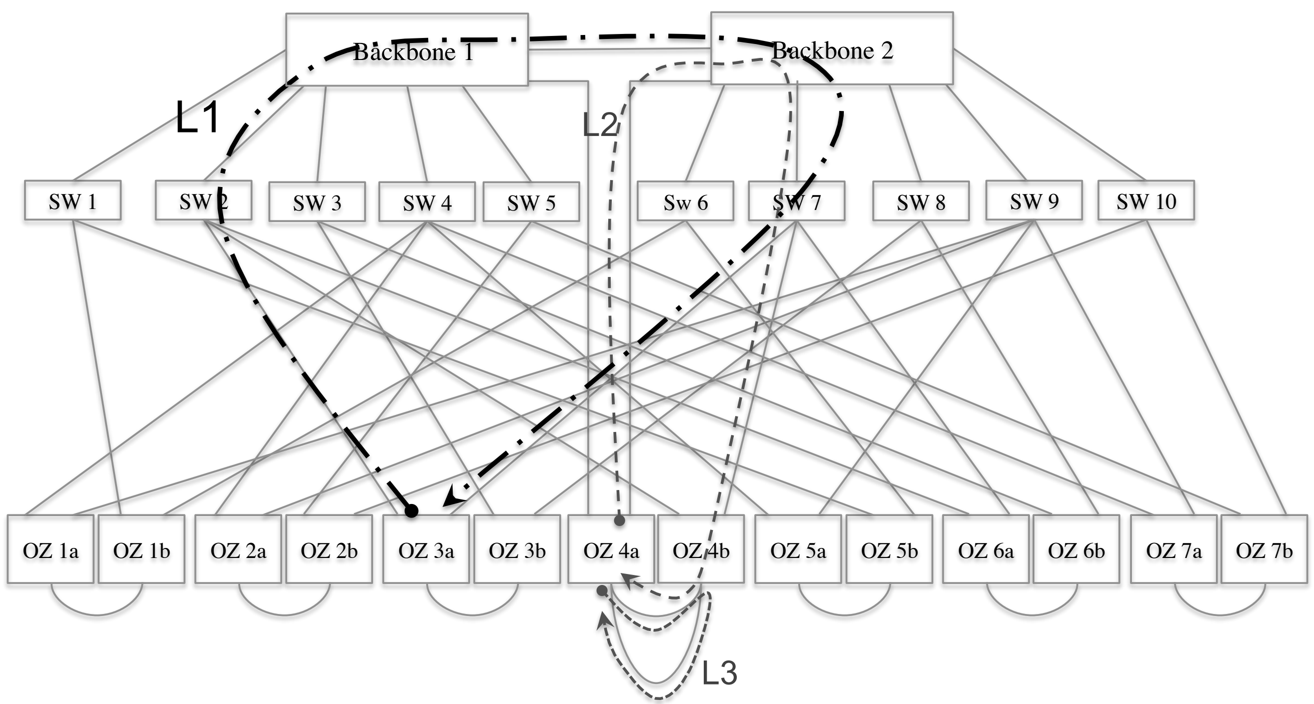
evaluation — slice isolation

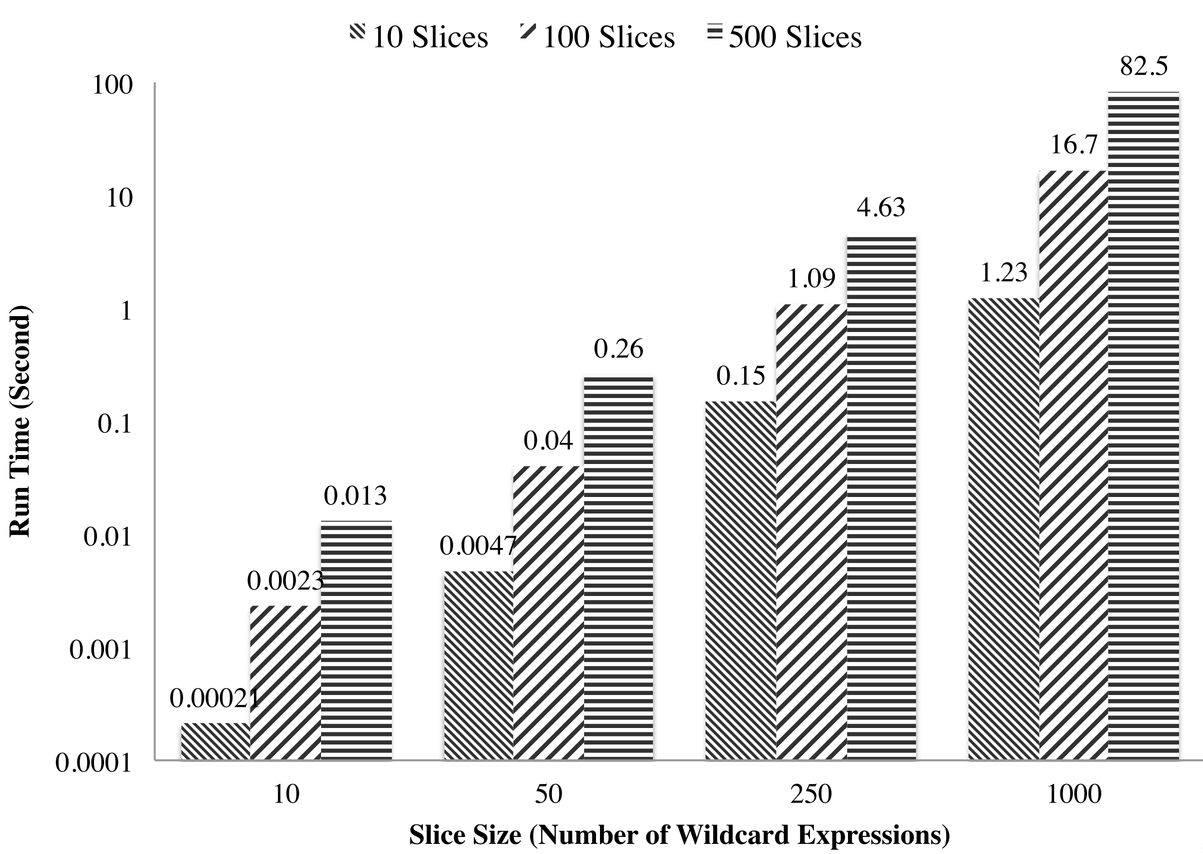




- check if a new slice is isolated from other slices at reservation time
  - randomly generate test slices

evaluation — slice isolation





- check if a new rewrite will leak packets (between slices)