Composing Middlebox and Traffic Engineering Policies in SDNs

Yiyang Chang*, Gustavo Petri†, Sanjay Rao*, and Tiark Rompf*
*Purdue University, †LIAFA — Université Paris Diderot

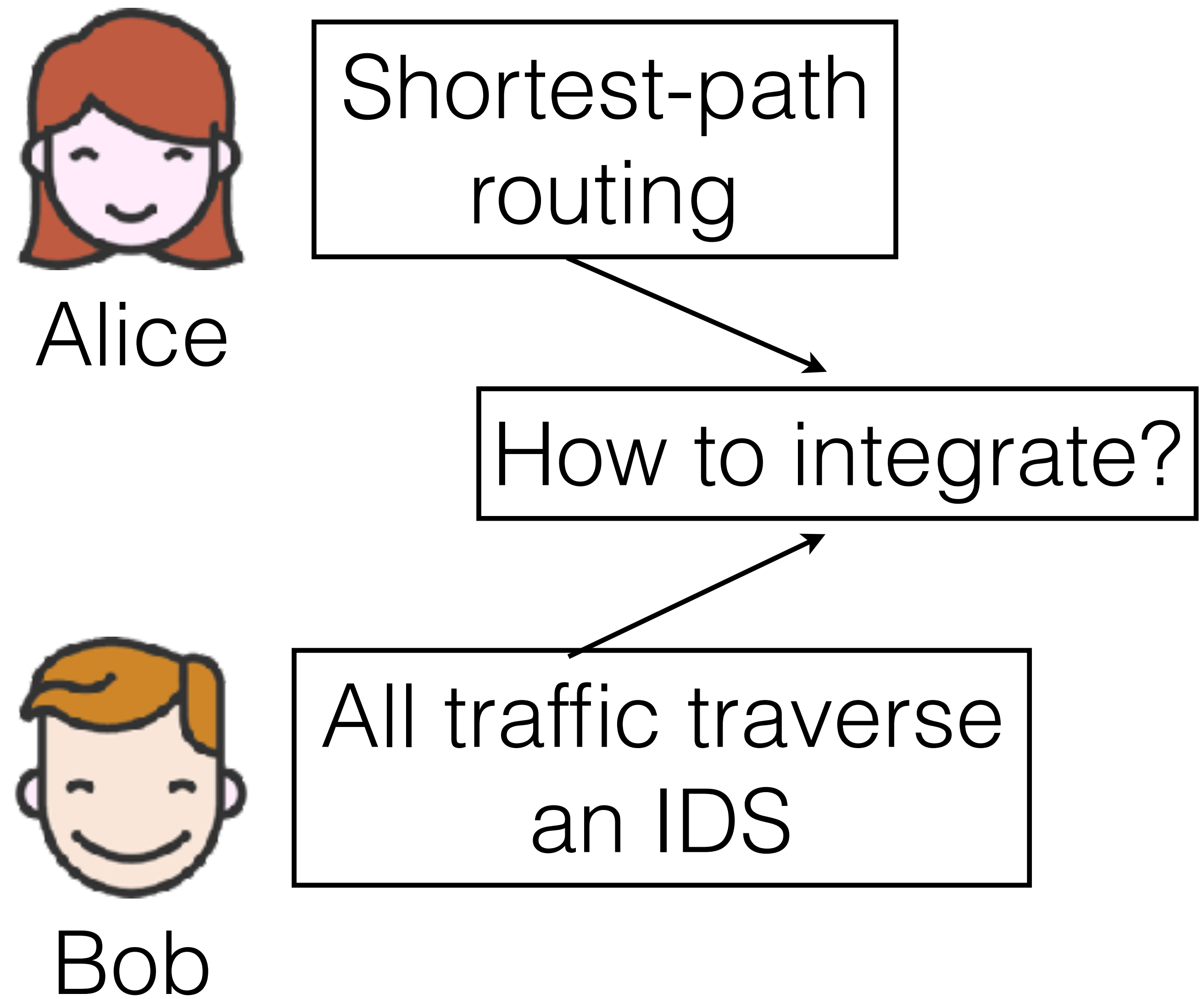
IEEE INFOCOM SWFAN 2017

Motivation

- Middlebox deployment is common in enterprise and ISP networks
    - Both capital cost and management cost are huge
- Different IT teams manage different classes of middleboxes
- How to integrate different requirements?

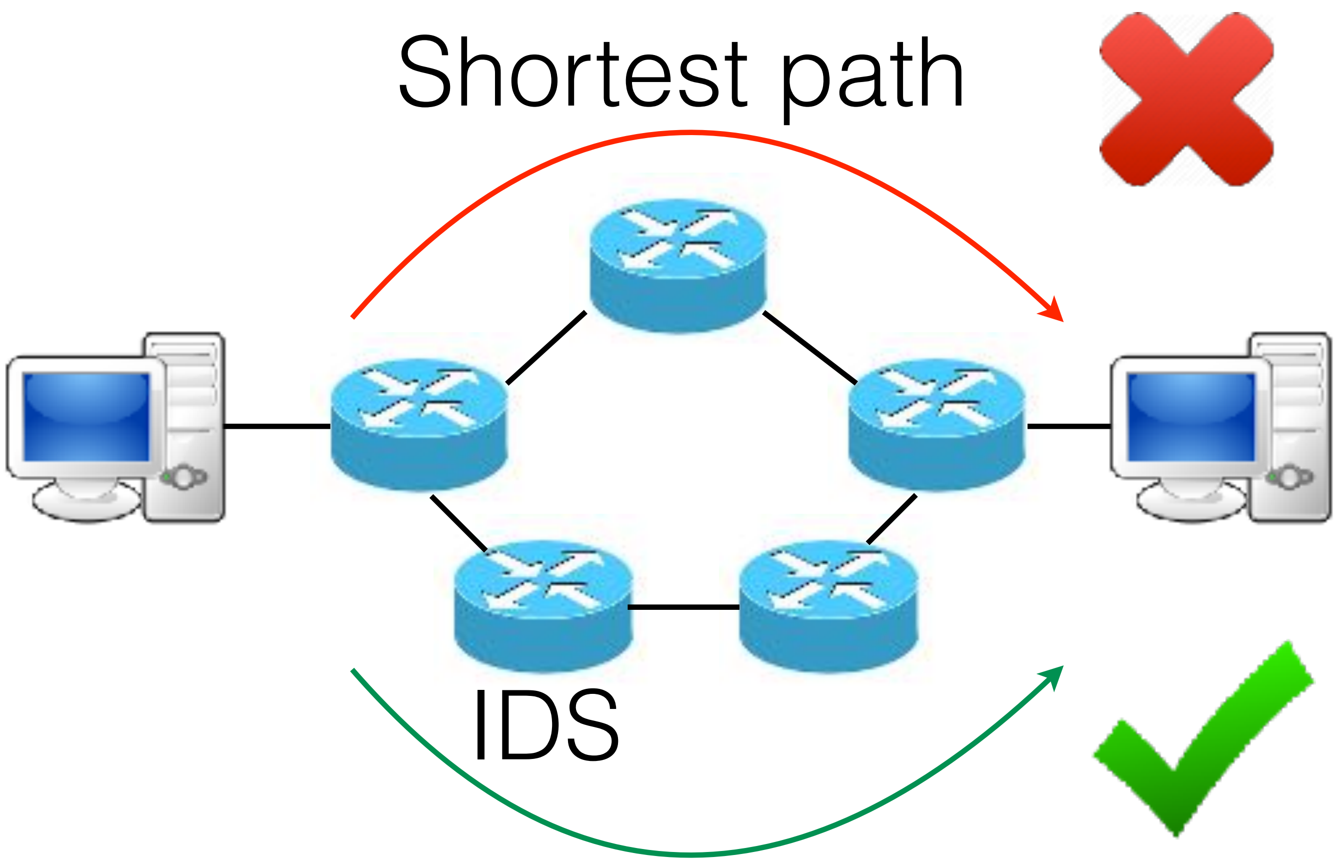
Composition is Non-trivial

- Alice manages routing module
    - Implements a shortest-path algorithm
- Bob manages IDS module
    - Enforces a policy that all traffic should traverse an IDS
- Could these modules be easily composed without Alice and Bob being explicitly aware of their respective implementations?



Why is Existing Solution Not Sufficient?

- Pyretic first computes paths in a general purpose language, and composition is done *after* generating the paths
- But, things can easily go wrong!
- Composition should be done *prior to* generating packet-forwarding policies



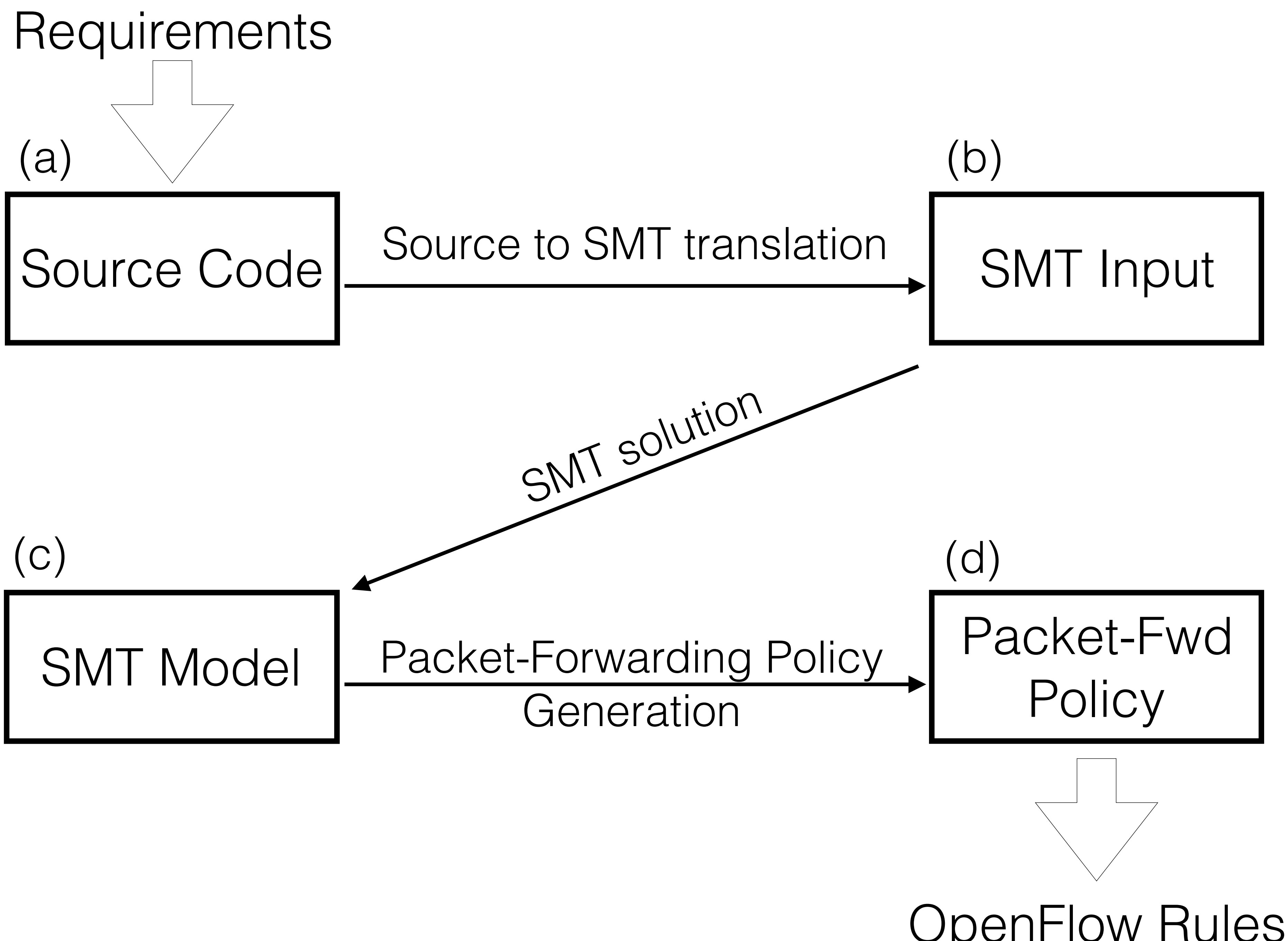
Our Solution

- We investigate an approach where compositionality is supported *prior to* the generation of packet forwarding policies

- Each application is written as a logic program, and provides a set of *requirements* that must be respected by a synthesized solution

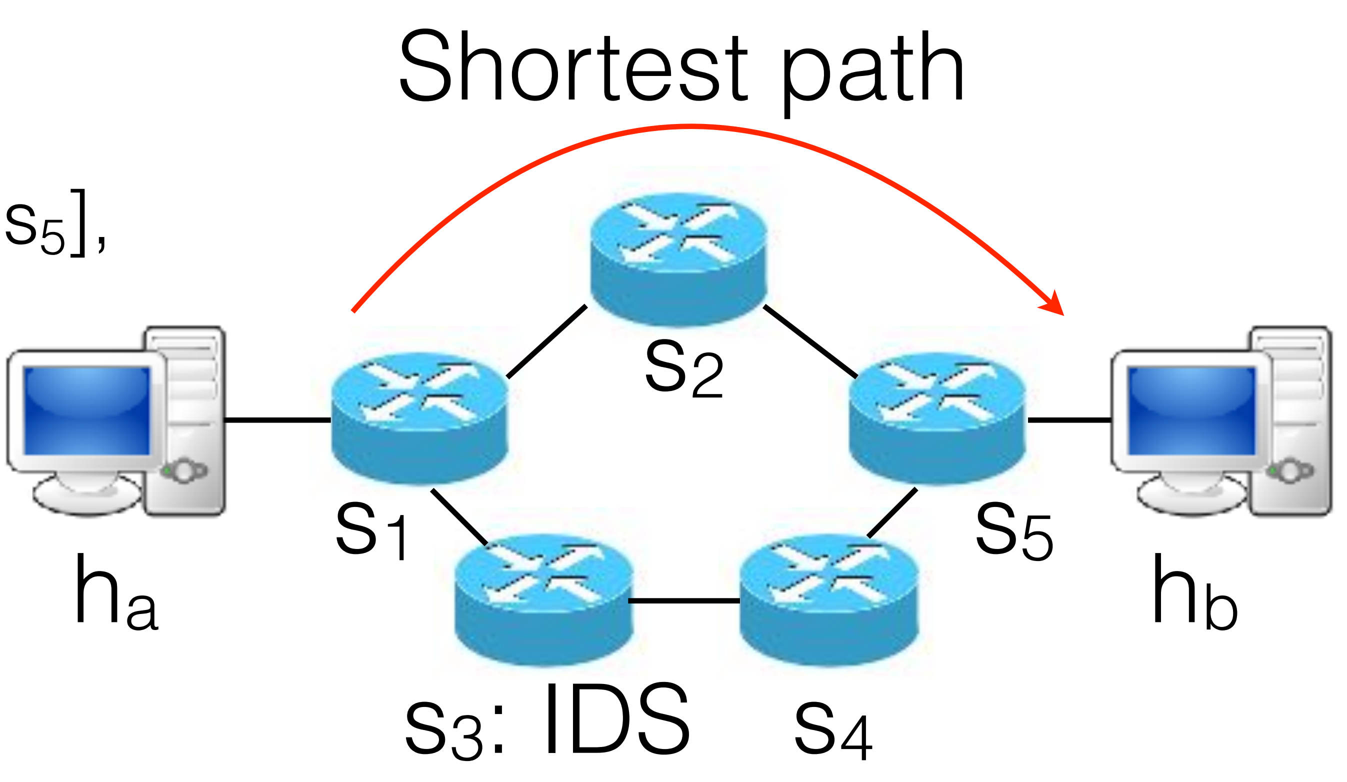
- A constraint solving engine iterates over these requirements to search the solution space and find a solution respecting all the requirements.

From Requirements to Rules



Composing Requirements - Revisit the Example

- Alice: Route from h_a to h_b
  - route(h_a, h_b, X)
  - Possible solution: X = [s_1, s_2, s_5], but fails to enforce IDS.



- Bob: All routes go through IDS
  - hasIDS([s_3 | X]).
    hasIDS([S | X]) :- hasIDS(X).
    routeIDS(h_a, h_b, X) :-
        route(h_a, h_b, X), hasIDS(X).
  - X = [s_1, s_3, s_4, s_5]

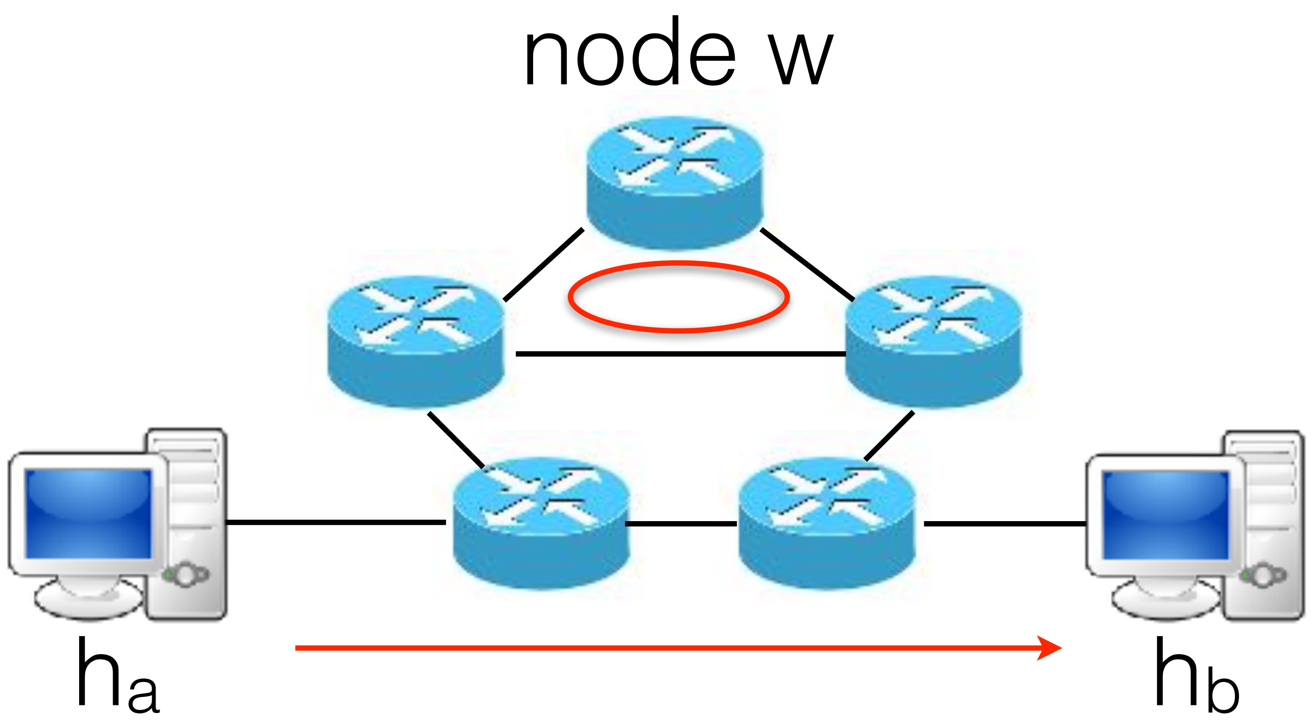
Translating Requirements to Constraints

- Naive composition may not work!
  - Classic shortest-path formulation (logic form)
    - $x_{i,j}$ = 1 if link <*i*, *j*> is in the path

$$(\exists i,\ x_{s,i}) \wedge (\exists i,\ x_{i,d})$$
$$\forall i, j,\ x_{i,j} \wedge (j \neq d) \Rightarrow \exists k,\ x_{j,k}$$
$$\forall i, j,\ x_{i,j} \wedge (i \neq s) \Rightarrow \exists k,\ x_{k,i}$$

    - Minimize the sum of all $x_{i,j}$
  - Add middlebox (node *w*) constraints

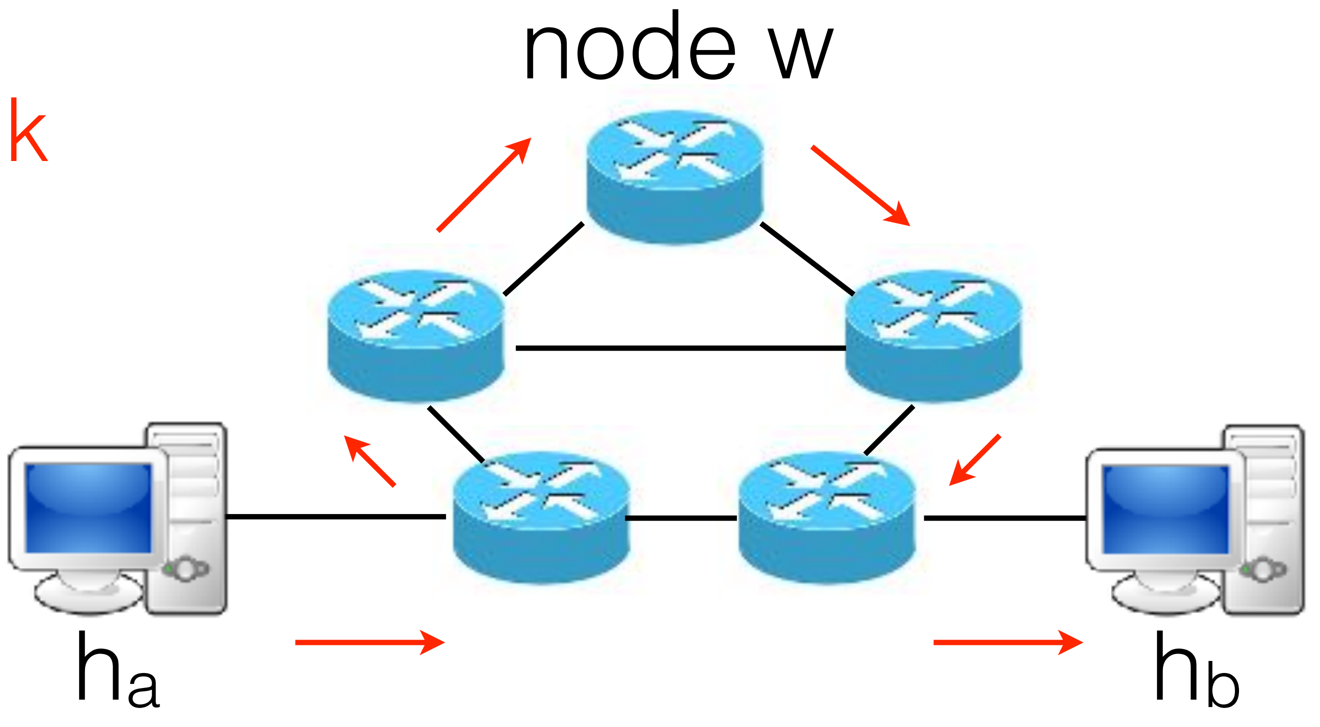
$$\exists j, x_{w,j}$$



- Solution contains a disconnected loop!
- We need a formulation supporting composition

Walk-based Shortest Path Formulation

- Walk-based shortest path formulation: Find a valid walk from a source node *s* to destination node *d*.
- Walk formulation explicitly prevents the disconnected loop
- Now safe for composition with middlebox requirements



Walk-based Shortest Path Formulation

$$x_{s,1} \wedge t_1$$

Source node s is scheduled first.

$$\forall i, k, \;\; x_{i,k} \wedge x_{j,k+1} \Rightarrow e_{i,j}$$

If node *i* is visited in step *k*, and *j* is visited in step *k + 1*, an edge must exist between nodes *i* and *j.*

$$\forall k, \;\; t_k \wedge \neg t_{k+1} \Rightarrow x_{d,k}$$

The last node of the walk is destination node *d*. The walk has exactly *k* steps.

$$\forall i, j, k, i \neq j \Rightarrow \neg x_{i,k} \vee \neg x_{j,k}$$

At most one node is visited in step *k*.

$$\forall i, k, \;\; x_{i,k} \Rightarrow t_k$$

If node *i* is visited in step *k*, the walk has at least *k* steps.

$$\forall k, \;\; \neg t_k \Rightarrow \neg t_{k+1}$$

$$\exists k, x_{d,k} \wedge \neg t_{k+1}$$

The destination node *d* exists in the path and eliminates trivial solutions.

Safely Composing Middlebox Requirements

- Translation of hasIDS()

$$\exists k, x_{w,k}$$

The node *w* must be traversed.

$$\exists k, w \in W, x_{w,k}$$

One of multiple IDS nodes in set *W* is traversed.

$$\exists k_1, k_2, x_{w_1,k_1} \wedge x_{w_2,k_2} \wedge (k_1 < k_2)$$

Node w_1 must be traversed prior to w_2.

More Composition Scenarios

- Bounding link utilization
- Multi-path routing
- Soft requirements to aid conflict resolution

Preliminary Results

- Path computation
  - Shortest-path
  - Shortest-path traversing a middlebox
- Implemented the walk-based formulation in Microsoft Z3 SMT solver (Python API)
- Evaluated with K-ary fat-tree topologies

Running time

- Running time of finding the shortest path, and the shortest path traversing one middlebox on different K-ary fat-trees
- The performance is acceptable for moderate-sized topologies.
    - Offline phase of traffic engineering
- Much room for performance improvement

K	# of nodes	Shortest-path (sec)	1-middlebox (sec)
4	20	0.08526	0.3298
8	80	2.226	11.94
12	180	40.67	262.6
16	320	285.3	725.2
20	500	2037	3978

Future Work

- Generality
    - Application beyond traffic engineering
- Performance
    - We demonstrated our framework with an SMT solver. It is interesting to explore the performance trade-offs with alternative solving engines, such as ILP solvers
- Source language
    - Current input language has a Prolog-like syntax
    - In the future we may consider a source level syntax more amenable to network operators such as a user defined syntax for relational operators.

Conclusions

- In this paper, we have explored how middlebox requirements may be incorporated in traffic engineering and SDN applications in a *compositional* manner.
- We have argued that doing so requires composition *prior to* the generation of packet-forwarding policies, in contrast to current approaches that perform composition *after* packet-forwarding policies are generated.

Thanks!

Questions?