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

Requirements

(a) Source Code

Source to SMT translation

(b) SMT Input

SMT solution

(c) SMT Model

Packet-Forwarding Policy Generation

(d) Packet-Fwd Policy

OpenFlow Rules
Composing Requirements - Revisit the Example

• Alice: Route from $h_a$ to $h_b$
  
  • $\text{route}(h_a, h_b, X)$
  
  • Possible solution: $X = [s_1, s_2, s_5]$, but fails to enforce IDS.

• Bob: All routes go through IDS
  
  • $\text{hasIDS}([s_3 \mid X])$.
    $\text{hasIDS}([S \mid X]) :\text{- hasIDS}(X)$.
    $\text{routeIDS}(h_a, h_b, X)$ :-
      $\text{route}(h_a, h_b, X)$, $\text{hasIDS}(X)$.

• $X = [s_1, s_3, s_4, s_5]$
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
      
      \[
      (\exists i, x_{s,i}) \land (\exists i, x_{i,d})
      \land 
      \forall i,j, x_{i,j} \land (j \neq d) \Rightarrow \exists k, x_{j,k}
      \land 
      \forall i,j, x_{i,j} \land (i \neq s) \Rightarrow \exists k, x_{k,i}
      \]
    - 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

- Source node $s$ is scheduled first.
- If node $i$ is visited in step $k$, and $j$ is visited in step $k + 1$, an edge must exist between nodes $i$ and $j$.
- The last node of the walk is destination node $d$. The walk has exactly $k$ steps.
- At most one node is visited in step $k$.
- If node $i$ is visited in step $k$, the walk has at least $k$ steps.
- The destination node $d$ exists in the path and eliminates trivial solutions.
Safely Composing Middlebox Requirements

- Translation of \texttt{hasIDS()}

\[
\exists k, x_w, k \\
\exists k, w \in W, x_w, k \\
\exists k_1, k_2, x_{w_1, k_1} \land x_{w_2, k_2} \land (k_1 < k_2)
\]

- The node \( w \) must be traversed.
- One of multiple IDS nodes in set \( W \) is traversed.
- 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

<table>
<thead>
<tr>
<th>K</th>
<th># of nodes</th>
<th>Shortest-path (sec)</th>
<th>1-middlebox (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20</td>
<td>0.08526</td>
<td>0.3298</td>
</tr>
<tr>
<td>8</td>
<td>80</td>
<td>2.226</td>
<td>11.94</td>
</tr>
<tr>
<td>12</td>
<td>180</td>
<td>40.67</td>
<td>262.6</td>
</tr>
<tr>
<td>16</td>
<td>320</td>
<td>285.3</td>
<td>725.2</td>
</tr>
<tr>
<td>20</td>
<td>500</td>
<td>2037</td>
<td>3978</td>
</tr>
</tbody>
</table>
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?