NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion
I shouldn’t be the one giving this talk…

Third year PhD student @ETH Zürich
Papers at NSDI, SIGCOMM, PLDI, CAV, SOSR, …
Check him out at  hassany.ps
NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion

Ahmed El-Hassany
Petar Tsankov
Laurent Vanbever
Martin Vechev
Curious if the Internet is also better during IETF/NANOG/RIPE...

Fewer heart attack patients die when top cardiologists are away at conferences, study finds

Heart attack patients are more likely to survive when top cardiologists are not in the hospital, a new study suggests. Researchers at Harvard Medical School...
Curious if the Internet is also better during IETF/NANOG/RIPE...

Fewer heart attack patients die when top cardiologists are away at conferences, study finds

Heart attack patients are more likely to survive when top cardiologists are not in the hospital, a new study suggests. Researchers at Harvard Medical School...
Curious if the Internet is also better during IETF/NANOG/RIPE...

Fewer heart attack patients die when top cardiologists are away at conferences, study finds
Heart attack patients are more likely to survive when top cardiologists are not in the hospital, a new study suggests. Researchers at Harvard Medical School...
Yes.
Yes.
The Internet seems to be better off during week-ends…

% of route leaks
source: Job Snijders (NTT)
“Human factors are responsible for 50% to 80% of network outages”

Juniper Networks, What’s Behind Network Downtime?, 2008
Google routing blunder sent Japan's Internet dark on Friday

Another big BGP blunder

By Richard Chirgwin 27 Aug 2017 at 22:35

Last Friday, someone in Google fat-thumbed a border gateway protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole.

The trouble began when The Chocolate Factory "leaked" a big route table to Verizon, the result of which was traffic from Japanese giants like NTT and KDDI was sent to Google on the expectation it would be treated as transit.

Since Google doesn't provide transit services, as BGP Mon explains, that traffic either filled a link beyond its capacity, or hit an access control list, and disappeared.

The outage in Japan only lasted a couple of hours, but was so severe that Japan Times reports the country's Internal Affairs and Communications ministry wants carriers to report on what went wrong.

BGP Mon dissect what went wrong here, reporting that more than
In August 2017

Someone in Google fat-thumbed a Border Gateway Protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole.
In August 2017

Someone in Google fat-thumbed a Border Gateway Protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole.

[...] Traffic from Japanese giants like NTT and KDDI was sent to Google on the expectation it would be treated as transit.
In August 2017

Someone in Google fat-thumbed a Border Gateway Protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole.

[...] Traffic from Japanese giants like NTT and KDDI was sent to Google on the expectation it would be treated as transit.

The outage in Japan *only lasted a couple of hours* but was so severe that [...] the country's Internal Affairs and Communications ministries want carriers to report on what went wrong.
Configuration synthesis addresses this problem by deriving low-level configurations from high-level requirements.
Configuration synthesis addresses this problem by deriving low-level configurations from high-level requirements.

Inputs

Network model

Physical topology

High-level requirements given by the operator

Outputs

```
router ospf 1
router-id 120.1.7.7
redistribute bgp 700 subnets
!
router bgp 700
neighbor 125.1.17.1 remote-as 100
!
address-family ipv4
 redistribute ospf 1 match internal external 1 external 2 neighbor 125.1.17.1 activate
!
address-family ipv4 multicast
 network 125.1.79.0 mask 255.255.255.0
 redistribute ospf 1 match internal external 1 external 2 neighbor 125.1.17.1 activate
!
```

Configuration synthesis addresses this problem by deriving low-level configurations from high-level requirements given by the operator.
Configuration synthesis: a booming research area!

Out of high-level requirements, automatically derive...

- **Genesis** [POPL’17] forwarding rules
- **Propane** [SIGCOMM’16] BGP configurations
- **PropaneAT** [PLDI’17] BGP configurations
- **SyNET** [CAV’17] OSPF + BGP configurations
- **Zeppelin** [SIGMETRICS’18]
Synthesizing configuration is great, but comes with challenges preventing a wide adoption
Existing synthesizers...
Problem #1 interpretability can produce configurations that widely differ from humanly-generated ones
Problem #1  interpretability  can produce configurations that widely differ from humanly-generated ones

Problem #2  continuity  can produce widely different configurations given slightly different requirements

Existing synthesizers...
Problem #1 interpretability can produce configurations that widely differ from humanly-generated ones

Problem #2 continuity can produce widely different configurations given slightly different requirements

Problem #3 deployability cannot flexibly adapt to operational requirements, requiring configuration heterogeneity

Existing synthesizers...
A key issue is that synthesizers do not provide operators with a fine-grained control over the synthesized configurations.
Introducing...

NetComplete
NetComplete allows network operators to flexibly express their intents through **configuration sketches**

A configuration with “holes”
interface TenGigabitEthernet1/1/1
  ip address ? ?
  ip ospf cost 10 < ? < 100

router ospf 100
  ...

router bgp 6500
  ...
  neighbor AS200 import route-map imp-p1
  neighbor AS200 export route-map exp-p1
  ...
  ip community-list C1 permit ?
  ip community-list C2 permit ?

route-map imp-p1 permit 10
  ...
  match community C2

route-map exp-p1 ? 10
  match community C2
  route-map exp-p2 ? 20
  match community C1
  ...
  route-map exp-p2 ? 20
  match community C1
interface TenGigabitEthernet1/1/1
  ip address ??
  ip ospf cost 10 < ? < 100

router ospf 100
  
  ...

router bgp 6500
  
  ...
  neighbor AS200 import route-map imp-p1
  neighbor AS200 export route-map exp-p1
  ...
  ip community-list C1 permit ?
  ip community-list C2 permit ?

Holes can identify specific attributes such as:

- IP addresses
- link costs
- BGP local preferences
interface TenGigabitEthernet1/1/1
  ip address ??
  ip ospf cost 10 < ? < 100

router ospf 100
  ...

router bgp 6500
  ...
  neighbor AS200 import route-map imp-p1
  neighbor AS200 export route-map exp-p1
  ...
  ip community-list C1 permit ?
  ip community-list C2 permit ?

route-map imp-p1 permit 10
  ...

route-map exp-p1 ? 10
  match community C2
  route-map exp-p2 ? 20
  match community C1
  ...

Holes can also identify entire pieces of the configuration
NetComplete “autocompletes” the holes such that the output configuration complies with the requirements.
interface TenGigabitEthernet1/1/1
ip address ??
ip ospf cost 10 < ? < 100

router ospf 100
...

router bgp 6500
...
neighbor AS200 import route-map imp-p1
neighbor AS200 export route-map exp-p1
...
ip community-list C1 permit ?
ip community-list C2 permit ?

route-map imp-p1 permit 10
route-map exp-p1 ? 10
  match community C2
route-map exp-p2 ? 20
  match community C1
...
interface TenGigabitEthernet1/1/1
  ip address 10.0.0.1 255.255.255.254
  ip ospf cost 15

router ospf 100
  network 10.0.0.1 0.0.0.1 area 0.0.0.0

router bgp 6500
  ... 
  neighbor AS200 import route-map imp-p1
  neighbor AS200 export route-map exp-p1
  ... 
  ip community-list C1 permit 6500:1
  ip community-list C2 permit 6500:2

route-map imp-p1 permit 10
  set community 6500:1
  set local-pref 50

route-map exp-p1 permit 10
  match community C2

route-map exp-p2 deny 20
  match community C1
  ...
NetComplete reduces the autocompletion problem to a constraint satisfaction problem.
<table>
<thead>
<tr>
<th>First</th>
<th>Encode the</th>
<th>protocol semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>high-level requirements</td>
<td>as a logical formula (in SMT)</td>
</tr>
<tr>
<td></td>
<td>partial configurations</td>
<td></td>
</tr>
</tbody>
</table>
First Encode the

- high-level requirements as a logical formula (in SMT)
- partial configurations

Then Use a solver (Z3) to find an assignment for the undefined configuration variables s.t. the formula evaluates to True
Main challenge:

**Scalability**

**Insight #1**

- **network-specific heuristics**
- search space navigation

**Insight #2**

- **partial evaluation**
- search space reduction
NetComplete: **Practical Network-Wide Configuration Synthesis with Autocompletion**

1. BGP synthesis
   - optimized encoding
2. OSPF synthesis
   - counter-examples-based
3. Evaluation
   - flexible, *yet* scalable
NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion

1. BGP synthesis
   optimized encoding

   OSPF synthesis
   counter-examples-based

Evaluation
flexible, yet scalable
NetComplete autocompletes router-level BGP policies by encoding the desired BGP behavior as a logical formula.
M ⊨ Reqsp BGP_{protocol} ∧ Policies
$M \models \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies}$

how should the network forward traffic
concrete, part of the input
M ⊨ \textbf{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies}

R1.\text{BGP}_{\text{select}}(A1,A2) \land 
R1.\text{BGP}_{\text{select}}(A2,A3) \land \ldots
Concrete, protocol semantic

How do BGP routers select routes

\[ M \models \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies} \]
\[ \text{BGP}_{\text{select}}(X,Y) \leftrightarrow (X.\text{LocalPref} > Y.\text{LocalPref}) \lor \ldots \]

\[ M \models \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies} \]
how routes should be modified
symbolic, to be found

M ⊨ Reqs \land \text{BGP}_{\text{protocol}} \land \text{Policies}
M ⊨ \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies}

R1.\text{SetLocalPref}(A1) = \text{VarX} \\
R1.\text{SetLocalPref}(A2) = 200
Solving this logical formula consists in assigning each symbolic variable with a concrete value

\[ \text{BGP}_{\text{select}}(X,Y) \leftrightarrow (X.\text{LocalPref} > Y.\text{LocalPref}) \lor \ldots \]

\[ \mathbf{M} \models \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies} \]

\[ \text{R1.} \text{BGP}_{\text{select}}(A_1,A_2) \land \quad \text{R1.} \text{SetLocalPref}(A_1) = \text{VarX} \]
\[ \text{R1.} \text{BGP}_{\text{select}}(A_2,A_3) \land \ldots \quad \text{R1.} \text{SetLocalPref}(A_2) = 200 \]
\[ \mathcal{M} \models \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies} \]

\[
\begin{align*}
\text{BGP}_{\text{select}}(X, Y) & \iff (X.\text{LocalPref} > Y.\text{LocalPref}) \lor \ldots \\
\text{R1.} & \text{BGP}_{\text{select}}(A1, A2) \land \\
\text{R1.} & \text{BGP}_{\text{select}}(A2, A3) \land \ldots \\
\text{R1.} & \text{SetLocalPref}(A1) = \text{VarX} \\
\text{R1.} & \text{SetLocalPref}(A2) = 200
\end{align*}
\]
\( \text{VarX} := 250 \quad M \models \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies} \)

\[
\begin{align*}
\text{BGP}_{\text{select}}(X, Y) &\iff (X.\text{LocalPref} > Y.\text{LocalPref}) \lor \ldots \\
\text{R1}.\text{BGP}_{\text{select}}(A_1, A_2) &\land \\
\text{R1}.\text{BGP}_{\text{select}}(A_2, A_3) &\land \ldots \\
\text{R1}.\text{SetLocalPref}(A_1) &= \text{VarX} \\
\text{R1}.\text{SetLocalPref}(A_2) &= 200
\end{align*}
\]
Naive encodings lead to complex constraints that cannot be solved in a reasonable time
Naive encodings lead to complex constraints that cannot be solved in a reasonable time.

\[ M \models \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies} \]

- challenges
- BGP x OSPF
- huge search space
Naive encodings lead to complex constraints that cannot be solved in a reasonable time.

\[
M \models \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies}
\]

- challenges
- solutions
- BGP x OSPF
- iterative synthesis
- huge search space
- partial evaluation
Naive encodings lead to complex constraints that cannot be solved in a reasonable time.

\[ M \models \text{Reqs} \land \text{BGP}_{\text{protocol}} \land \text{Policies} \]

- Challenges: huge search space
- Solutions: iterative synthesis, partial evaluation
- Solutions: BGP x OSPF
NetComplete encodes reduced policies by relying on the requirements and the sketches
NetComplete encodes reduced policies by relying on the requirements and the sketches.

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Capture how announcements should propagate using the requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>BGP propagation graph</td>
</tr>
</tbody>
</table>
NetComplete encodes reduced policies by relying on the requirements and the sketches

Step 1  Capture how announcements should propagate using the requirements
Output  BGP propagation graph

Step 2  Combine the graph with constraints imposed by sketches via symbolic execution
Output  partially evaluated formulas
NetComplete relies on the requirements to figure out where BGP announcements should (not) propagate
NetComplete relies on the requirements to figure out where BGP announcements should (not) propagate

Requirement
Only customers should be able to send traffic to Provider #2
NetComplete relies on the requirements to figure out where BGP announcements should (not) propagate.

Requirement

Only customers should be able to send traffic to Provider #2.
NetComplete computes one BGP propagation graph per equivalence class
NetComplete concretizes symbolic announcements by propagating them through the graph and sketches.

Encode BGP policies as SMT formulas.

For all ann in Announcements:
  ann.communities = [External, Var1]
  ann.local_pref = 100

Inject symbolic announcement

permitted = True
local_pref = ?
communities = ?
...

Result is a partially evaluated formula

Cust

permitted = True
local_pref = 100
communities = [External, Var1]
...

...
NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion

BGP synthesis
optimized encoding

OSPF synthesis
counter-examples-based

Evaluation
flexible, yet scalable
As for BGP, Netcomplete phrases the problem of finding weights as a constraint satisfaction problem.
Consider this initial configuration in which the (A,C) traffic is forwarded along the direct link
For performance reasons, the operators want to enable load-balancing
What should be the weights for this to happen?
input requirements

synthesis procedure
∀ \( X \in \text{Paths}(A,C) \setminus \text{Reqs} \)

\[ \text{Cost}(A \to C) = \text{Cost}(A \to D \to C) < \text{Cost}(X) \]
∀X ∈ Paths(A,C)\Reqs

Cost(A→C) = Cost(A→D→C) < Cost(X)

Solve
∀X ∈ Paths(A,C)\Reqs

Cost(A→C) = Cost(A→D→C) < Cost(X)

Solve
∀X ∈ Paths(A,C)\Reqs

Cost(A→C) = Cost(A→D→C) < Cost(X)

Solve
This was easy, but... it does not scale

\[ \forall X \in \text{Paths}(A,C) \setminus \text{Reqs} \]

\[ \text{Cost}(A \rightarrow C) = \text{Cost}(A \rightarrow D \rightarrow C) < \text{Cost}(X) \]

Solve
There can be an exponential number of paths between A and C...

\( \forall X \in \text{Paths}(A, C) \) \( \text{Reqs} \)

\[ \text{Cost}(A \rightarrow C) = \text{Cost}(A \rightarrow D \rightarrow C) < \text{Cost}(X) \]

Solve
To scale, NetComplete leverages **Counter-Example Guided Inductive Synthesis (CEGIS)**

An contemporary approach to synthesis where a solution is iteratively learned from counter-examples
While enumerating all paths is hard, computing shortest paths given weights is easy!
input requirements
input requirements

synthesis procedure
∀ \( X \in \text{SamplePaths}(A,C) \setminus \text{Reqs} \)
∀X ∈ SamplePaths(A,C)\Reqs
Sample: { [A,B,D,C] }
∀X ∈ SamplePaths(A,C) \ Reqs

Cost(A→C) = Cost(A→D→C) < Cost(X)
∀ \( X \in \text{SamplePaths}(A,C) \setminus \text{Reqs} \)

\[ \text{Cost}(A \rightarrow C) = \text{Cost}(A \rightarrow D \rightarrow C) < \text{Cost}(X) \]

Solve
∀X ∈ SamplePaths(A,C) \ Reqs

Cost(A → C) = Cost(A → D → C) < Cost(X)

Solve
∀ \( X \in \text{SamplePaths}(A,C) \backslash \text{Reqs} \)

Cost(\( A \rightarrow C \)) = Cost(\( A \rightarrow D \rightarrow C \)) < Cost(\( X \))

Solve

Synthesized weights
The synthesized weights are incorrect:
\[ \text{cost}(A \rightarrow B \rightarrow C) = 250 < \text{cost}(A \rightarrow C) = 300 \]
We simply add the counter example to SamplePaths and repeat the procedure
The entire procedure usually converges in few iterations making it very fast in practice
NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion

BGP synthesis
optimized encoding

OSPF synthesis
counter-examples-based

Evaluation
flexible, yet scalable
Question #1

Can NetComplete synthesize large-scale configurations?

Question #2

How does the concreteness of the sketch influence the running time?
We fully implemented NetComplete and showed its practicality

<table>
<thead>
<tr>
<th>Code</th>
<th>~10K lines of Python</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SMT-LIB v2 and Z3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>OSPF, BGP, static routes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>as partial and concrete configs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Cisco-compatible configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>validated with actual Cisco routers</td>
</tr>
</tbody>
</table>
## Methodology

| Topology          | 15 topologies from Topology Zoo  
small, medium, and large |
|-------------------|--------------------------------------------------------------------------------|
| Requirement       | Simple, Any, ECMP, and ordered (random)  
using OSPF/BGP          |
| Sketch            | Built from a fully concrete configuration  
from which we made a % of the variables symbolic |
NetComplete synthesizes configurations for large networks in few minutes
NetComplete synthesizes configurations for large networks in few minutes.

<table>
<thead>
<tr>
<th>Network size</th>
<th>Reqs. type</th>
<th>Synthesis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>Simple</td>
<td>14s</td>
</tr>
<tr>
<td>~150 nodes</td>
<td>ECMP</td>
<td>13s</td>
</tr>
<tr>
<td></td>
<td>Ordered</td>
<td>249s</td>
</tr>
</tbody>
</table>

OSPF synthesis time (sec) settings
16 reqs, 50% symbolic, 5 repet.
CEGIS enabled
Without CEGIS, OSPF synthesis is >100x slower and often timeouts
NetComplete synthesis time increases as the sketch becomes more symbolic.
NetComplete synthesis time increases as the sketch becomes more symbolic.

OSPFF synthesis time (sec)

% of weights left symbolic in the sketch

settings

16 reqs
large topos.
NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion

BGP synthesis
optimized encoding

OSPF synthesis
counter-examples-based

Evaluation
flexible, yet scalable
NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion

Autocompletes configurations with “holes”
leaving the concrete parts intact

Phrases the problem as constraints satisfaction
scales using network-specific heuristics & partial evaluation

Scales to realistic network size
synthesizes configurations for large network in minutes
NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion

Ahmed El-Hassany
Petar Tsankov
Laurent Vanbever
Martin Vechev