Verification and Configuration of Software-based Networks

Serena Spinoso
XXIX PhD cycle
serena.spinoso@polito.it
Advisor: Prof. Riccardo Sisto

29 June 2017
New opportunities in productive environments, like data centers, thanks to:

- **Network Function Virtualization** (NFV) decouples software implementation of the network functions from their physical counterparts

- **Software Defined Networking** (SDN) is in charge of chaining those functions to create network paths.
**Scenario**

Service Providers allow users to build **Service Graphs** by:

- selecting a set of **Virtual Network Functions** (VNFs)
  - DPI, NAT, Firewall ...

- specifying traffic forwarding through the selected VNFs
  - **Service Function Chaining** (SFC)
Thesis Goals

Many research topics in SDN/NFV-based networks:
- VNF placement, security enforcement, bandwidth optimization, ...

**PhD goals:**
1. Formal verification of service graph requests
2. Model-based configuration of network functions
Formal Verification of Service Graphs
Formal verification checks the correctness of computer systems before putting them into use, by exploiting formal methods and mathematical reasoning.

Formal verification can be applied to the networking field:

- **Network Verification** proves that a network model (e.g. network configurations) fulfills certain invariants.
Formal Network Verification

Formal techniques for verification and property checking in *SDN/NFV-based networks* to avoid faults and errors at run-time

- flexibility of the offered network services
- very frequent network reconfiguration (e.g. user requests or management events)

*Traditional model checking techniques run out of memory and time in case of complex network scenarios!*
Formal Network Verification: Challenges

Service Providers need verification strategies:

- done *before* deploying the service graphs
- with *low* verification times
- with *fair* processing resources (e.g. CPU, memory)
Formal Verification of Service Graphs

- Detecting Anomalies in Service Function Chains
- Checking Reachability in Service Graphs
Detecting Anomalies in SFCs

Many SDN programming languages offer **forwarding policies**
- to specify traffic forwarding through chains of VNFs
- translated into flow entries in OpenFlow switches

Providers have to check the policy specification correctness
- faults in network configurations may arise at run-time
Detecting Anomalies in SFCs: contributions

State of the Art:
- verification of OpenFlow networks
  - *during or after* the switch configuration deployment
  - just *conflict analysis* among OpenFlow entries

Contributions:
- language-independent checking mechanism
- early-detection of anomalies among forwarding policies
- customizable anomaly specification approach
Detecting Anomalies in SFCs: anomalies

**Forwarding anomalies** are any erroneous or unwanted policy specification done by the Service Provider

- possible faulty network conditions and states at run-time
- include conflicts, errors, sub-optimization, and more

Example:

- “FW must process traffic before NAT"
- “User1 traffic must not pass through FW"
Detecting Anomalies in SFCs: the approach

**First Order Logic models** for:

- Forwarding rules that compose a policy

\[
    r_i = (M_i, C_i) = (eth_{src} = v_{eth_{src}}, \ eth_{dst} = v_{eth_{dst}}, \ 
    eth_{type} = v_{eth_{type}}, \ vlan_{id} = v_{vlan_{id}}, \ ip_{src} = v_{ip_{src}}, \ 
    ip_{dst} = v_{ip_{dst}}, \ ip_{proto} = v_{ip_{proto}}, \ port_{src} = v_{port_{src}}, \ 
    port_{dst} = v_{port_{dst}}, \ c_1^i, \ldots, c_n^i), \ r_i \in \mathbb{R}_F
\]

- **Pre-** and **provider-defined** anomalies

\[
    ip_{src} = ip_{src_j} \land ip_{dst} = ip_{dst_j} \land ip_{proto} = ip_{proto_j} \land \\
    port_{src} = port_{src_j} \land port_{dst} = port_{dst_j} \land c_1^i \neq c_1^j \Rightarrow \text{Collision}(pi_i, pi_j)
\]
Detecting Anomalies in SFCs: classification

Figure: Hierarchy of anomaly classes.
Detecting Anomalies in SFCs: results

Verification time is influenced by both the *number* of forwarding rules and the *percentage* of these that trigger an anomaly.

*Reasonable verification times from both the NFV architecture and users perspectives!*

---

**Graphs:**

- **Left graph:** Verification time vs. number of forwarding rules for different percentages of anomalies (10%, 20%, 50%, 80%)
- **Right graph:** Verification time vs. percentage of anomalous forwarding rules for different numbers of forwarding rules (100, 300, 500, 700, 1000)
Detecting Anomalies in SFCs: publications and next plans

Publications:


Next improvements:

- raise the abstraction-level of traffic flow modelling
- raise the expressiveness of the model
  - additional network operation (e.g. monitoring)
Formal Verification of Service Graphs

- Detecting Anomalies in Service Function Chains
- Checking Reachability in Service Graphs
Checking Reachability in Service Graphs

Forwarding errors may also be due to faulty VNF configurations at run-time

- e.g. faulty filtering rules in firewalls, wrong black list in anti-spams, DPIs etc...

Service Providers need more accurate modelling approaches to check the network correctness
Checking Reachability in SGs: contributions

State of the Art:
- most existing verification tools are OpenFlow-oriented
- they check network functions that take forwarding decisions based on packet headers only (stateless functions)

Contributions:
- Model networks and stateful VNFs
  - network functions that alter and forward packets based on internal states and algorithms
Checking Reachability in SGs: the approach

Boolean modelling and satisfiability checking techniques to verify reachability properties against stateful VNFs

VeriGraph:

- Network and VNFs models are sets of FOL formulas
- Use Z3, an SMT solver, as verification engine
- Exploit Neo4J for service graph manipulation
Checking Reachability in SGs: network model

FOL formulas for modelling:

1. **network fundamentals**

\[(send(n_0, n_1, p_0, t_0)) \implies (n_0 \neq n_1 \land p_0.\text{src} \neq p_0.\text{dst}), \quad \forall n_0, p_0, t_0\]

2. **VNF behaviour** (e.g. NAT)

\[(send(nat, n_1, p_1, t_1) \land \neg isPrivateAddress(p_1.\text{dst})) \implies p_1.\text{src} = \text{ip}_\text{nat} \]
\[\land \exists (n_0, p_0, t_0) \mid (t_0 < t_1 \land recv(n_0, nat, p_0, t_0) \land isPrivateAddress(p_0.\text{src}) \land p_0.\text{dst} = p_1.\text{dst}), \quad \forall (n_1, p_1, t_1)\]
Checking Reachability in SGs: VeriGraph

Part of the SP-DevOps toolkit (UNIFY) and integrated into a VNF Orchestrator (ESCAPE)

- [https://github.com/netgroup-polito/verigraph](https://github.com/netgroup-polito/verigraph)

Part of the D-release of Parser (OPNFV)

- [https://github.com/opnfv/parser](https://github.com/opnfv/parser)
## Checking Reachability in SGs: VeriGraph

<table>
<thead>
<tr>
<th>Reach. Property</th>
<th>Result</th>
<th>VeriGraph</th>
<th>Z3</th>
<th>Verification Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host1 -&gt; Server</td>
<td>X</td>
<td>263ms</td>
<td>30ms</td>
<td>293ms</td>
</tr>
<tr>
<td>Host2 -&gt; Server</td>
<td>X</td>
<td>326ms</td>
<td>28ms</td>
<td>354ms</td>
</tr>
<tr>
<td>Host3 -&gt; Server</td>
<td>V</td>
<td>256ms</td>
<td>54ms</td>
<td>310ms</td>
</tr>
<tr>
<td>Host1 -&gt; Server</td>
<td>V</td>
<td>250ms</td>
<td>107ms</td>
<td>357ms</td>
</tr>
<tr>
<td>Host2 -&gt; Server</td>
<td>V</td>
<td>295ms</td>
<td>65ms</td>
<td>360ms</td>
</tr>
<tr>
<td>Host3 -&gt; Server</td>
<td>V</td>
<td>282ms</td>
<td>61ms</td>
<td>343ms</td>
</tr>
</tbody>
</table>

### Diagram

![Network Diagram](image)
Checking Reachability in SGs: publications and next plans

Publications:


Next improvements:

- Make the verification approach scalable
- Reduce the complexity of the modelling technique
Checking Reachability in SGs: scalability issues

Boolean modelling-based approach is more promising than model checking techniques

- FOL is not a decidable logic

Make network and VNF models in **Skolemized form** (without existential quantifiers)
Network Function Configuration
Service Providers have to **configure** VNFs to complete the service graph deployment
- filtering rules in firewalls, IP addresses for NATs...

Cloud Managers (CMs) rely on external configuration services
- e.g. Puppet, Chef, Ansible,...
Network Function Configuration: Challenges

Flexible configuration approaches have to consider:

- **many configuration strategies** per function
  - REST API, CLI, SMTP, etc...

- **configuration semantics** depend on the **function types**
  - router and firewall parameters are clearly different
Network Function Configuration

- Seamless Configuration of Virtual Network Functions
Seamless Configuration of VNFs: contributions

State of the art:
- existing configuration services are targeted to expert users
- they use one configuration strategy
- they rely on VNF-specific plug-ins

Contributions:
- Enable a **model-based VNF configuration** in CMs to:
  - hide the low-level details to the users
  - support *many* configuration strategies
  - use *vendor-agnostic* and *function-independent* modules
Seamless Configuration of VNFs: overview

Service Providers need configuration modules that are *vendor-agnostic* and *function-independent*:

- **Translator**
  - translates the configuration parameters into a particular format required by a VNF

- **Gateway**
  - delivers the produced configuration into the VNF
Seamless Configuration of VNFs: inputs

**VNF Object Model** (VNF OM)
- representation of the main VNF configuration parameters
- one VNF Object Model Instance is created for each VNF

**Translation Rules**
- directives to translate the VNF OM instance into the structure/format required by the VNF

**Access Parameters**
- directives to push down the VNF configuration
Two types of VNF were successfully configured in software cloud architecture provided by **PLUMgrid, Inc**:  

![Graph showing performance results of Vyatta Core and Bind9](image-url)
Seamless Configuration of VNFs: publications and next plans

Publications:


**Next improvements:**

- integration with the verification service to
  - check the correctness of the VNF configuration generated
  - enable an automatic fixing in case of errors
Conclusions

Improvements to the state of the art in many aspects of a network service life-cycle:

- Verification of anomalies in a forwarding policy
- Reachability analysis in service graphs
- Model-based functional configuration of network functions
Publication List

2014:

2015:
Publication List

2016:

2017:
- Spinoso S., Sisto R., Formal Verification of Forwarding Policies. Submitted major revision to: Transactions on Network and Service Management, IEEE.

Planned:
Questions