Service Provider DevOps

Wolfgang John (Ericsson Research) and the UNIFY WP4 team
UNIFYing cloud and carrier networks

› Increased velocity of service introduction

› Unified network-cloud programming abstraction and orchestration

› Novel, integrated management features to coping with high service velocity
DevOps in the IT and DC world

• An agile approach to deliver software, with the goals of:
  – Increased deployment frequency & faster TTM
  – Lower failure rate
  – Faster mean time to recovery
  – Maximum predictability, efficiency, and maintainability of operational processes
  – Programmable and automatic processes

Service Provider DevOps

DevOps is to a large degree a question of culture
Research in “organizational behavior” out of scope in this project
Focus is on supporting DevOps principles from a technical perspective

Four DevOps principles [1]...
• Develop and test against production-like systems
• Deploy with repeatable, reliable processes
• Monitor and validate operational quality
• Amplify feedback loops

...translated to UNIFY SP-DevOps:
• Supporting service and VNF development with sandboxed environments
• Automation of deployment, with capabilities for continuous verification and observability
• Verification and observability, supporting automatic troubleshooting workflows
• Expose verification, observability and troubleshooting functions through programmatic APIs

SP DevOps vs DC DevOps

• SP environment vs. Datacenter environments
  – Higher spatial distribution– telecom resources are spread over wide areas for coverage
  – Heterogeneous hardware and software
  – Lower levels of redundancy in access and aggregation networks compared to typical data centers
  – Carrier-grade requirements on high availability and latency
  – Likely to have business boundaries between Dev and Op roles, keeping some “silos” intact
UNIFY SP-DEVOPS CONCEPT

- Troubleshooting
- Observability
- Operator
- Verification
- Deploy
- Define
- Code

- Service developer VNF Developer
- VNF developer support
SP-DEVOPS CHALLENGES

• Main challenges identified in UNIFY
  – Observability:
    • Scalable and resource-efficient monitoring and support functions
    • Automatic programmability of observability capabilities
  – Verification
    • Automatic deploy-time verification of service definitions and configurations, to identify problems early in the lifecycle
    • Run-time verification of configurations
  – Troubleshooting:
    • Automatic workflows for a dynamic environment
    • Monitoring and debugging information accessible through common interfaces
UNIFY SP-DevOps toolkit

Bottom-up approach:

> UNIFY is supporting SP-DevOps with

> 9 tools and

> 4 supporting functions

In the following, we present some selected examples of UNIFY SP-DevOps tools based on maturity of the solution!
UNIFY Architecture

Service model verification

AutoTPG - Run-time flow verification

EPOXIDE - Multicomponent debugging tool

Observability:
- EWSDN '15 paper: W. John et al., “Scalable Software defined monitoring for SP-DevOps”
- EWSDN’15 Demo: F. Moradi et al., “Monitoring Transport and Cloud for NFV”
- Demo at IM 2015 [3]

Service Model Verification

Problem:
Checking the NF-FG network configuration against desired policies during all steps of deployment (i.e. fast!)
Verification ensures consistency and security
Example of properties:
- connectivity from node A to node C
- isolation from node B to node D

For this, we need to formally model:
- Every NF that takes part to the NF-FG to be deployed
  - Even VNFs that alter packets (e.g. NAT, VPN GW, etc.)
- The whole network including the forwarding strategies of each function and the different links connecting them (derived from the NF-FG + configuration)

Solution:
We are extending an already existing tool, based on logic formulas representing the VNFs behavior and their interconnections, in order to model the network.

How does it work?
- We extract all the possible chains from the NF-FG
- We feed Z3 (a SOTA SMT solver) with our input (the logic formulas) to produce a result:
  - All the supported policies are converted into reachability tests between two network nodes;
  - Z3 tells if the problem is SATisfied (reachability granted) or UNSATisfied (nodes are isolated)
- We provide the verification result to the orchestrator

Evaluation:
First Performance results are promising:
- Each reachability test runs in less than 150ms
- The verification time is less than 50ms in most cases

**AutoTPG: Run-time flow verification**

**Problem:**
- Verification of the Flow-Matching Functionality of OpenFlow switches for matching errors (i.e. to find all the packet headers that cannot be matched)
- Existing tools (most notably ATPG) do not verify the data plane functionality for matching-header related issues. Wildcarded flow entries are only tested with one packet header, the rest remains untested.
- **Testing all header possibilities, while minimizing additional bandwidth requirements**

**Solution:**
- AutoTPG transmits test packets to verify the Flow-Matching functionality.
- To overcome the bandwidth challenge, test packets are forwarded using duplicated Flow Entries (copied)
  - Duplicated Flow Entries either drop or forward the test packets to the controller
  - Assumption: if the matching error is present in the original Flow Entry, the matching issue is also present in the copied Flow Entries

**Steps of Verification**
1. Flow Entries Duplication in another FlowTable
2. Test Packet Generation Step
   - Packet-Out messages
3. Matching Error identification step
   a. Binary Search Method (on flow counters)
   b. Packet Reception Method (on controller)

Pan-European network layout. Five Flow Entries per switch, the source and dst IP address contain 24 bits mask length, and five different packet headers cannot be matched correctly with a Flow Entry (matching error)

**Problem:**
- Series of hypothesis testing to find a bug / misconfiguration / faulty element.
- SDN: mixture of distributed heterogeneous HW & SW components
  - Specific tools for specific tasks
  - Difficult to interconnect them
- New SDN application -> new troubleshooting challenge

**Solution:**
A troubleshooting framework to make the combination of special-purpose tools more effective

**Epoxide** is an Emacs-based modular framework to effectively combine existing troubleshooting tools:
- Simplifies troubleshooting processes
- Inspection of intermediate steps
- Saving/replaying/sharing of Troubleshooting graphs
- Integration of existing tools with text input/output
- Collection of new node implementations by 3rd-parties

**Why Emacs?**
- It's an extensible text-editor, lots of add-on packages
  - Buffers represent textual data
  - Editable source files
  - Interactive shells
  - Output of sub processes
  - Easy navigation among buffers

---

**Troubleshooting Graphs** as the main EPOXIDE building blocks

- Nodes and links of TS Graphs are shown in buffers
  - Allowing semantic navigation
  - Navigation in a visualized TS graph
- Real-time, interactive extension of TS graphs
- TS graph is written in a simple click-inspired language

**Event driven framework**
**API for node developers**

**VNF Development support:**
- Modify SG to include parallel service chain with duplicated traffic
- Potentially compare results
- Debug new NF (NF2') in production env.
- Attach EPOXIDE to SAP2 for debugging
Roles in UNIFY

Service Model Verification supports:
* Verification processes by functional verification of abstract service models (SG, NF-FG + configurations) throughout the deploy-time in real-time.

AutoTPG supports:
* Verification processes by Run-time verification of flow entries in the network
* Observability and Troubleshooting processes by active measurements support root-case analysis

EPOXIDE supports
* Troubleshooting processes by offering a generic SDN debugging tool
* VNF developers by facilitating observability, verification and troubleshooting capabilities
SP-DevOps: Summary

• The SP-DevOps concept* as a framework pursuing DevOps goals for Telco services
• Three tools presented above, contributing to Verification, Troubleshooting, and VNF Development support
• Two additional tools to be presented at EWSDN, contribution to observability
• Next steps:
  • Programmability of observability capabilities
  • Automatic Troubleshooting workflows
  • SP-DevOps Toolkit v1 release planned for early November

Key take-aways

- The SP-DevOps supports DevOps principles for Telco services with technical processes:
  - Scalable and efficient **observability** for continuous service and infrastructure monitoring
  - Fast **verification** of complex service graphs during deploy-time
  - A framework facilitating **troubleshooting** of service graphs and VNFs for both operators and developers
  - A set of tools solving specific challenges within the processes, many of them to be published open-source (**SP-DevOps Toolkit**)
Backup Slides
Roles identified

– Dev A: Service Developer (definition of the Service Graph)
  Dev B: VNF Developer (software developer of new VNFs)
– Op: Operator (ensuring performance indicators of a service)

Identified four groups of technical functionality in focus

– Verification
  • Automated verification of service definitions and configurations during deploy and run-time
– Observability
  • Providing visibility onto the operational performance of service graphs deployed
– Troubleshooting
  • Isolating the cause of an unexpected behaviour
  • Automated usage of verification and observability mechanisms
– VNF development support
  • Deploy and test VNFs in an isolated slice of the production environment
SP-DEVOPS CONCEPT (2)

- “Define” service graph definition by a service developer
  - “Code” VNF software by VNF Developer
- “Deploy” represents fulfillment functions
  - “isolated deploy” in the execution environment for VNF development
- “Operate” with assurance functions for production environment and VNF
  - “SW debugging” for the VNF Developer in a realistic unified production environment.

“VNF transition” to production environment by making it available to the “Define” stage of SP-DevOps once the software debugging was successful.
Service Model Verification

Problem statement

• While deploying an NF-FG, we need an automatic tool to check its network configuration against desired policies before it is deployed.

• Example of properties:
  – connectivity from node A to node C
  – isolation from node B to node D

• Each modification of the graph triggers a new verification phase in order to continuously ensure consistency and security

• We need to formally model:
  – Every NF that takes part to the NF-FG to be deployed
  – The whole network including all the forwarding strategies of each function and the different links connecting them.
  – The properties we want to check.
The approach: model checking with Z3

We are extending an already existing tool, based on logic formulas representing the VNFs behavior and their interconnections, in order to model the network.

How it works?

• We extract all the possible chains from the NF-FG description
• We feed Z3 (a SOTA SMT solver) with our input (the logic formulas) to produce a result:
  – All the supported policies are converted into reachability tests between two network nodes;
  – Z3 tells if the problem is SATisfied (reachability granted) or UNSATisfied (nodes are isolated)
• We provide the verification result to the higher levels (e.g. the orchestration level)
Evaluation results

• First results on a simple scenario

Performance results are promising:
• Each reachability test runs in less than 150ms
• The verification time is less than 50ms in most cases
Role in UNIFY

Service Model Verification supports
Verification processes by functional verification of abstract service models (SG, NF-FG + configurations) throughout the deploy-time in real-time.
Run-time flow verification

AutoTPG tool

- AutoTPG verifies the Flow-Matching Functionality of OpenFlow switches for matching errors
  - It finds the packet headers that cannot be matched correctly with the Flow-Match Header of a Flow Entry

- The verification of the Flow-Match functionality is difficult by just analyzing the configuration of networks
  - AutoTPG transmits test packets to verify the Flow-Matching functionality.

- The closely related tool is the automatic test packet generation (ATPG) tool
  - However, this tool does not verify the data plane functionality for matching-header related issues.
Challenges and method of Verification

- If test packets have to match with the Flow Entries of data packets, these would need additional bandwidth to be reserved for test packets on the links corresponding to the outgoing actions of the matched Flow Entries
  - To overcome the challenge, we forward the test packets using duplicated Flow Entries (copied)
    - Duplicated Flow Entries either drop or forward the test packets to the controller
    - The assumption is that if the matching error is present in the original Flow Entry, the matching issue is also present in the copied Flow Entries

- Steps of Verification
  - Flow Entries Duplication in another FlowTable
  - Test Packet Generation Step
    - Packet-Out messages
  - Matching Error identification step
    - Binary Search Method
      - It reads counters of Flow Entries and applies the binary search algorithm to find errors
    - Packet Reception Method
      - The controller receives the test packets back and from the un-received packets, the controller finds the errors
Results: Verification time vs bandwidth available

The verification time is the time required to find the matching errors presents in the Flow Entries.

Five Flow Entries per switch, the source and destination IP address contain 24 bits mask length, and five different packet headers cannot be matched correctly with a Flow Entry (matching error).
Software or hardware errors

1. Here, errors mean which packet-headers can not be matched correctly with the Flow-Match Header of Flow Entries

Functional block of OpenFlow switches such as NEC, HP
Role in UNIFY

AutoTPG supports:
Verification processes by Run-time verification of flow entries in the network
Observability and Troubleshooting processes by active measurements support root-case analysis

Multicomponent debugging tool


*Code available as Opensource: https://github.com/nemethf/epoxide
Debugging / Troubleshooting in SDN

Series of hypothesis testing to find a software bug / mis-configuration / faulty element.

- SDN: mixture of distributed heterogeneous HW & SW components
- New SDN application → new troubleshooting challenge

Proposed approach (EPOXIDE): make the combination of special-purpose tools more effective by a troubleshooting framework
Epoxide mapping to the UNIFY architecture

Orchestration and Service Layers

Infrastructure controller

Infrastructure Domain

SAP

Source IP

Destination IP

Link ID

Loss monitor

Iperf

Split output

Resource mapping

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)

Mon. Func. (MF)
Epoxide: an Emacs-based prototype

Emacs
- extensible text-editor
- **Buffers**: represent textual data
  - Editable source files
  - Interactive shells
  - Output of sub processes
  - ...
- Easy navigation among buffers
- Lots of add-on packages

Epoxide
- Nodes and links of TSG are shown in buffers
  - semantic navigation
  - navigation in a visualized TSG
- Event driven framework
- API for node developers
- Real-time, interactive extension of TSG
- TSG is written in a simple DSL
Epoxide summary and next steps

- Epoxide simplifies troubleshooting process
- Intermediate steps can be inspected
- Troubleshooting graphs can be saved, replayed, shared
- Existing tools with text input/output can be integrated
- New node implementations by third-parties
  - collected in a node repository

- Source code: http://github.com/nemethf/epoxide

Possible future directions
- More wrappers around available tools
- General purpose “decision nodes”
  - for complete automation
- Complex TSGs for real-life scenarios
  - E.g., for home PC connectivity problems

What we need:
- engineers’ opinions
EPOXIDE - example

Figure 4: An example troubleshooting configuration and the corresponding Emacs buffers.
Role in UNIFY

EPOXIDE supports
Trouble shooting processes by offering a genericSDN debugging tool
VNF development support by facilitating observability, verification and troubleshooting capabilities