Automated Configuration and Measurement of Emulated Networks with AutoNetkit

Simon Knight
University of Adelaide/Cisco Systems
simon.knight@adelaide.edu.au

1. ABSTRACT

Emulated networks enable educators, researchers, and operators to conduct realistic network scenarios on commodity hardware. However each network device must be configured, typically in a low-level syntax. This time-consuming and error-prone process limits scalability and discourages repeated experimentation.

This demonstration will show a platform to automate emulated network configuration and measurement, making large-scale network experimentation accessible.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network Management*

Keywords

Emulation; Configuration Management

2. INTRODUCTION

Data networks are complex to configure, as high-level design goals must be implemented on a per-device basis using vendor-specific syntax. In today's service provider and enterprise networks, 'intelligent' provisioning systems are required to address challenge of the per-device syntax vagaries.

Network emulation, where virtual machines running real router software are connected, enables both researchers and network operators to study the configuration change effects, before they are deployed to production networks. Network emulation systems are available for both Open-Source, such as Netkit [7], and commercial routing platforms [1] [2] [3]. While emulation doesn't offer the per-packet performance fidelity of production networks, it offers a realistic implementation of a vendor's control-plane behaviour, complete with standards interpretations or software bugs. This allows more realistic network behaviour than simulating an algorithm, but without the resource requirements of constructing a hardware testbed.

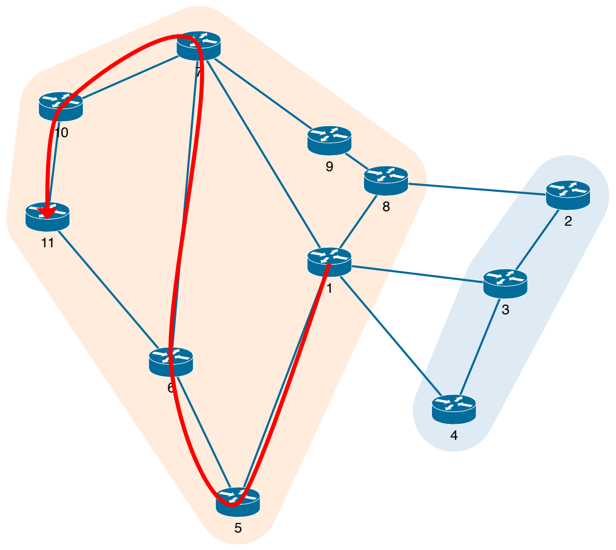

Figure 1: A two-AS Physical topology plotted using the visualisation system. Routers are grouped by their ASN attribute, and the parsed results of running `traceroute` on the emulated network are shown as a path overlay.

However, while emulation provides a platform for conducting such experiments, the experiments themselves must be constructed. This encounters the same complexities as configuring a production data network, where the translation from high-level to low-level is typically performed either manually by a network operator, or with the assistance of custom scripts. This manual or semi-automated practice is time consuming and error-prone for production networks, and limiting for conducting repeated experiments, where changing an input variable can require repeating the configuration process.

Recently, Software Defined Networking has received much research attention. One its core contributions is developing an abstract representation of networks. This decouples the network design and service topology from the low-level device configuration details, and enables programmatic network design, testing, and verification.

This system follows a similar approach, by providing an abstract network representation that allows high-level configuration of existing network devices. This enables configurations to be built from a succinct high-level description, with user-defined design rules allowing complicated custom topologies to be created in a high-level programming language. A real-time visualisation framework displays the topologies that result from the design rules, providing the user with verification of network designs before they are deployed. Decoupling network design from configuration generation allows different target platform configurations to be generated independent of the topology design.

To enable experimentation, this system provides a framework for automated deployment to emulation platforms, such as Netkit, and the ability to collect and process measurement data from the emulated network. By comparing the measured data to the original network design, it enables powerful closed-loop network experimentation and verification.

3. SYSTEM OVERVIEW

The current system is a rewrite of a previous prototype [4]. A Python API build on a graph-based core data structure has been added. The system is now modular and extensible through all of the measure modules, and the visualisation system and measurement parsing have been added.

It is written in Python, and can be installed on Windows, Linux, and Mac OS X, and is open-source, multi-platform, and scalable to networks with hundreds of nodes. It has been developed over a number of years [5], and is being used in a research and production capacity. It been used as the basis for large-scale emulated experiments [8] [6], in teaching and education, and to configure emulation platforms for large router vendors [1] [3]. It is hoped this demonstration stimulates development towards a common network configuration platform for educators, researchers, and operators.

4. WORKFLOW

Firstly, the input topology graph is drawn in a graphical editor such as yEd, with nodes representing routers, and edges the physical links connecting them. It is annotated with topology design attributes, such as the ASN value of a router, and the OSPF cost of a link. This graph is saved in GraphML format and read into AutoNetkit using NetworkX.

Pre-defined design rules, written in the Python API, are then applied to the input graph to create protocol topology graphs. These are constructed using the nodes, edges, and their attributes from the input graph, and can represent the topologies such as physical connectivity, OSPF, iBGP, eBGP, and IP addressing. A real-time visualisation system based on d3.js displays the design rules results, providing instant feedback on topology designs. As well as visual inspection, the API allows pre-deployment verification rules to be specified, ensuring network design constraints are observed.

These protocol-specific overlay graphs are condensed to a single device-level graph, using the compiler module. Each node contains the configuration attributes for to the device and associated operating system it represents, and the configuration template to render the node. This approach supports platform-specific configuration details, allowing configuration of multiple configuration languages. The render module pushes each device's configuration parameters into the specified template, to produce a set of device configuration files.

The deployment module packages and transfers these configuration files to the specified emulation server, and issues the relevant commands to start the emulation environment. We now have a running network emulation. The emulation size is only limited by the emulation server resources, allowing scalability to large network experiments. A user can log into the emulated routers and run commands such as `show ip route`.

Manually running measurement commands doesn't scale to large or repeated experiments. The system provides an API for the user to specify a measurement command, and the hosts to run it on, and automates the measurement process. The measured results are parsed into a Python data structure suitable for analysis and processing. The network topology constructed by the design rules can be analysed using graph algorithms such as *shortest path first*, and the output compared to the measured results. This provides verification that the network topology is running as designed. Finally, the results of these measurement and verification stages can be plotted in the visualisation system to visually confirm correct network behaviour.

5. DEMONSTRATION

The demonstration will show the workflow, from drawing to deployment and measurement, and the flexibility in modifying the components, such as the input topology and the network design rules, with the resulting topologies and measurement results shown in the real-time visualisation.

It will also show the live-network modification for OSPF weights, where a change in the input weights, or via the API results in the change being applied to the running network. The resulting IGP topology is then measured, compared to the theoretical topology, and the results displayed in the visualisation engine.

AutoNetkit is at `http://www.autonetkit.org`, the source code at `github.com/sk2/autonetkit`, and an example video at `http://youtu.be/Kyl_WpVJNVs`.

6. ACKNOWLEDGMENTS

The author would like to thank input and suggestions from Hung Nguyen, Iain Phillips, Olaf Maennel, Niklas Semmler, Joel Obstfeld, Randy Bush, Nick Falkner, and Matthew Roughan. This project was supported by the Australian Government through an Australian Postgraduate Award.

7. REFERENCES

[1] Cisco. Open Network Environment Webinar Series.

[2] GNS3. Graphical Network Simulator - GNS3.

[3] Juniper Networks, Inc. Junosphere User Guide.

[4] S. Knight, A. Jaboldinov, O. Maennel, I. Phillips, and M. Roughan. AutoNetkit: Simplifying Large Scale, Open-Source Network Experimentation. ACM Sigcomm Poster, 2012.

[5] H. Nguyen, M. Roughan, S. Knight, N. Falkner, O. Maennel, and R. Bush. How to Build Complex, Large-Scale Emulated Networks. *TridentCom*, 2011.

[6] I. Phillips, O. Maennel, D. Perouli, R. Austein, C. Pelsser, K. Shima, and R. Bush. RPKI propagation emulation measurement: an early report. IETF Talk, July 2012.

[7] M. Pizzonia and M. Rimondini. Netkit: easy emulation of complex networks on inexpensive hardware. In *Tridentcom 2008*.

[8] I. Poese, B. Frank, S. Knight, N. Semmler, and G. Smaragdakis. PaDIS emulator: An emulator to evaluate CDN-ISP collaboration. ACM Sigcomm Poster, 2012.