Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

TOPOGEN: A NETWORK TOPOLOGY GENERATION ARCHITECTURE WITH APPLICATION TO AUTOMATING SIMULATIONS OF SOFTWARE DEFINED NETWORKS

Andrés Laurito

Departamento de Computación
Facultad de Ciencias Exactas y Naturales (FCEyN)
Universidad de Buenos Aires (UBA)
Pab. 1, C1428EGA, Buenos Aires, ARGENTINA

Matías Bonaventura

Departamento de Computación
FCEyN, UBA and ICC, CONICET
Pab. 1, C1428EGA, Buenos Aires, ARGENTINA

Mikel Eukeni Pozo Astigarraga

European Laboratory for Particle Physics, CERN
Geneva 23, CH-1211
SWITZERLAND

Rodrigo Castro

Departamento de Computación
FCEyN, UBA and ICC, CONICET
Pab. 1, C1428EGA, Buenos Aires, ARGENTINA

ABSTRACT

Simulation is an important tool to validate the performance impact of control decisions in Software Defined Networks (SDN). Yet, the manual modeling of complex topologies that may change often during a design process can be a tedious error-prone task. We present TopoGen, a general purpose architecture and tool for systematic translation and generation of network topologies. TopoGen can be used to generate network simulation models automatically by querying information available at diverse sources, notably SDN controllers. The DEVS modeling and simulation framework facilitates a systematic translation of structured knowledge about a network topology into a formal modular and hierarchical coupling of preexisting or new models of network entities (physical or logical). TopoGen can be flexibly extended with new parsers and generators to grow its scope of applicability. This allows to design arbitrary workflows of topology transformations. We tested TopoGen in a network engineering project for the ATLAS detector at CERN.

1 INTRODUCTION

Operational computer networks are subjected to frequent reconfigurations in an effort to maintain their quality of service under uncertain conditions (produced by hardware, software or human failures). Meanwhile, the rapidly emerging Software Defined Networks (SDNs) technology offers an unprecedented capability to automatically and programmatically reconfigure large network topologies without the intervention of human operators. This new flexibility comes at the price of error proneness: the point of failure gets now shifted to the software that decides on network reconfiguration actions. The verification and validation of SDN-based design options becomes key to minimize the risk of deploying a faulty system.

Network simulation has long been an efficient tool to gain confidence with networks at design time. When network simulation models are built based on real, changing topologies, each modification implies the need for updating the simulation model accordingly.

The standard practice is to upgrade topology descriptions manually for a given modeling and simulation tool of choice. Such manual changes can get considerably time-consuming and error-prone, in particular for medium- to large-sized networks.

In this paper we introduce TopoGen, an SDN-oriented topology generator tool and reference architecture designed to create network simulation models automatically based on parseable network descriptions. Such descriptions rely on an intermediate topology abstraction that can be generated automatically, programmed manually from scratch or a mix of both (translated automatically first, tailored via programming later).

When dealing with the simulation of existing networks, TopoGen allows simulation models to keep up with frequent topology changes thanks to a Topology Intermediate Format (TIF) that can be automatically generated by querying real SDN controllers. For design phases where the real network does not exist yet, the network can also be prototyped using a Network Topology Model (NTM), an object-oriented network topology description language (based on Ruby). Thus, network designers do not need to be acquainted with the specifics of any particular network simulator.

NTM and TIF are two different pieces of the architecture, and none is intended to replace existing network description languages (in fact, such existing languages can take the role of input/output formats consumed/produced by TopoGen). We opted for defining our custom TIF to act as an in-memory relay format, independent of any third party existing language to describe networks. Regarding NTM, it belongs to the category of network specification languages, but has the salient feature that multi-hop flows can be described explicitly. Besides, NTM is native to TIF making the NTM Builder a trivial one-to-one object mapper. This sets an operational baseline that is agnostic of any third party language. Yet, the architecture leaves the door open for new Builders to be developed to accommodate known network description languages.

TopoGen can retrieve network topologies from different sources, in particular from SDN controllers such as the widely deployed ONOS (Berde et al. 2014) and OpenDayLight (Medved et al. 2014) implementations, and can generate simulation models for different outputs, notably DEVS-based simulators such as PowerDEVS (Bergero and Kofman 2011).

We show an application of TopoGen to a real-world network design project in the context of the Trigger and Data Acquisition (TDAQ) network of the ATLAS Experiment at CERN (ATLAS Collaboration 2008). A new network layer is added to a preexisting infrastructure, comprising approximately 120 nodes and 240 high speed links. TopoGen is first used to retrieve a candidate network topology prototyped by network engineers in Mininet (De Oliveira et al. 2014) connected to an ONOS SDN controller. Then, the topology is augmented with additional nodes to provide a more exhaustive representation of the future network whose performance is studied.

2 BACKGROUND

2.1 Software Defined Networking

Software Defined Networking (SDN) is an emerging architectural approach for computer networks where control logic is taken away from *switching devices* and moved up to centralized software running in *controller devices* (Kreutz et al. 2015).

In this new architecture, switching devices are lumped into much simpler packet forwarding elements operating at the so-called *Data Plane*. Controllers decide on and set up forwarding rules for each connected switching device, in an effort to comply with the overall quality of service requirements for the entire network. Controllers operate at the so-called *Control Plane* and concentrate most of the network service logic (e.g. monitoring, packet forwarding decisions -in case no rules are set up-, network topology discovery, etc.) Popular centralized SDN controller implementations (ONOS, OpenDayLight, Nox, Floodlight) provide different functionalities through APIs.

While the SDN approach promises boosted network flexibility, reconfigurability and scalability, the overall performance of the SDN main elements (the controller and its software components) is not yet clearly understood (Fernandes 2017). SDN performance is the motivation for active research to assess system limits (e.g. maximum forwarding rate and latency) recognizing that each controller implementation can perform very differently in different settings (Zhao et al. 2015). Moreover, even when correct at the

functional level, the control actions decided by an efficient controller may provoke undesired inefficiencies at the performance level in the underlying controlled network.

In this context, modeling and simulation-driven network engineering (Bonaventura et al. 2016) stands as a promising strategy: system verification with dynamic simulations can mitigate the risks of deploying functional SDN controllers that may cause poor quality of performance. Among the services implemented by SDN controllers, topology discovery keeps the topology information updated. Another service is the exposure of the known information about the network, usually by means of APIs (providing e.g. number of nodes, hosts, links, etc., and network metrics such as number of bytes forwarded per link, packet loss rates per port, etc.). TopoGen harnesses SDN services to systematically create up-to-date simulation models.

2.2 DEVS–Based Network Simulation With PowerDEVS

PowerDEVS (Bergero and Kofman 2011) is a discrete event simulator that implements the DEVS mathematical formalism (Zeigler et al. 2000) capable of representing any type of discrete system and approximating continuous systems with controlled accuracy. PowerDEVS provides a graphical interface to compose DEVS models via hierarchical block diagrams. While PowerDEVS can represent any kind of discrete-event system, it provides a model library specific to computer network simulation (Castro and Kofman 2015).

In PowerDEVS systems can be built by composing graphically pre-developed units of behaviour (*atomic models*) and structures (*coupled models*) from a model library (e.g. routers, switches, links, generators, etc.) and interconnecting them through input/output ports. In DEVS, structure and behaviour are kept under strict separation. The interconnection of several atomic and/or coupled models creates the *coupling information* and in our case matches the *network topologies* directly.

As with other simulators, defining large topologies graphically can be a tedious task. Vectorial DEVS (Bergero and Kofman 2014) makes it possible to graphically represent multiple instances, but only for regular topologies. Also, there is a possibility to program the network topology in C++ at the cost of higher code complexity and a detachment between code and graphical layouts.

We adopt the DEVS formalism and the PowerDEVS tool in our case study as they currently support the TDAQ network engineering team of the ATLAS experiment at CERN (Bonaventura et al. 2016, Foguelman et al. 2016), introduced below. Yet, the TopoGen architecture could be directly applied to produce network topologies for any other DEVS-based toolkits (Van Tendeloo and Vangheluwe 2017) that accepts a file-based structured specification of models (e.g. CD++ (Bonaventura et al. 2013) or VLE (Quesnel et al. 2009)).

3 RELATED WORK

There are several network simulators available both for commercial and academic use (Wehrle et al. 2010). They vary in several aspects: the discrete-event techniques and principles (sequential or parallel, replication- or decomposition-based, CPU- or GPU-based) (Ngangue Ndih and Cherkaoui 2015); the library of reusable models, and the software interfaces to assist the modeling activity (e.g. to define a network topology).

In some simulation packages network model behaviour and model topology are defined intermingled in the code (e.g. NS-3 (Carneiro 2010)). While this allows for great flexibility, the code can soon become too complex to understand, debug and maintain. A number of simulation tools (e.g. OPNET (Chang 1999), OMNET++ (Varga and Hornig 2008)) provide graphical editors which allow for an easy and compact understanding of the network topology, separating topology from model behaviour.

Nevertheless, defining a topology graphically can soon become inflexible for mid- to large-sized topologies (adding thousands of nodes with drag and drop methods can be very tedious and time-consuming). To address this issue some tools combine graphical editors with domain-specific languages (e.g. OMNET++) making it possible to parametrize the number of nodes and use programming-like structures to describe regular interconnect structures. This approach is efficient to describe large, mostly regular, topologies, but presents some limitations: 1) the modeler learns a description language that is specific to a single simulation

tool, 2) a new topology always needs to be created from scratch, and 3) when dealing with an existing network, there is no guarantee that a network description accurately represents the real system.

The alternative presented in this paper tends to mitigate these problems by accommodating all the aforementioned methods under a common architecture to define/transform network topologies: either using a graphical editor (when available), programming code (when desired), or using automatic data retrieval (e.g. from SDN controllers, if needed).

Meanwhile, network description languages exist beyond the simulation domain. For example, YANG (Bjorklund 2010) relies on an XML oriented approach. VXDL (Koslovski et al. 2008) allows to specify virtual resources, interconnections, topology, etc. in great detail. NDL (Van der Ham et al. 2007) can describe optical networks and is used by applications to query network capabilities to perform requests. The Internet Topology Zoo (ITZ) (Knight et al. 2011) provides a wide range of real Internet topologies in GML format (Himsolt 2010). Different software toolkits (including simulators) can find some languages more suitable than others to retrieve network information. For example in (Großmann and Schuberth 2013) a GML parser was implemented to generate Mininet models out of ITZ topologies.

Parsers and generators typically serve for a specific application and are usually coupled together. A parser can not be reused to generate other formats and a generator can not be used with other input formats.

In this work we introduce an intermediate network format (a core piece of the architecture) to act as a bridge between network definition languages and software tools willing to consume those descriptions. To the best of our knowledge there exist no equivalent solutions that permit flexible and extensible translation of various sources into various targets.

4 MOTIVATING CASE STUDY: DESIGNING THE FELIX NETWORK AT CERN

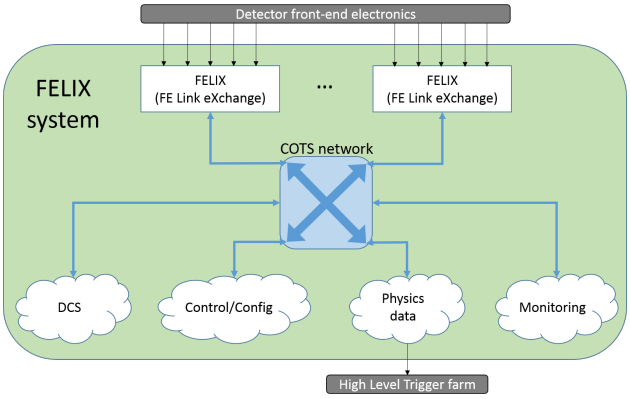


Figure 1: FELIX system components.

The ATLAS experiment at CERN hosts one of the four detectors at the Large Hadron Collider (LHC) where bunches of particles collide every 25 ns. Currently, the ATLAS detector generates information at about 80 TB/s which needs to be filtered before it can be permanently stored for offline analysis. The TDAQ layered system reduces a 40 MHz collision event rate down to 1 kHz by analyzing events in real time. A first-level trigger (L1) uses custom electronics, filtering events down to roughly 100 kHz. L1-accepted events are temporarily transfered over custom optical point-to-point fibers to 100 Read-Out System (ROS) server nodes. The High Level Trigger (HLT) accesses events stored in the ROS to further filter the data by running selection algorithms on approximately 2000 server nodes interconnected with 1 Gbps and 10 Gbps Ethernet links.

For 2025, the ATLAS experiment is planning full deployment of the new Front-End LInk eXchange (FELIX) system (Anderson et al. 2015), shown in Figure 1, that aims at interfacing between detector electronics and the TDAQ system. FELIX is meant to replace the custom point-to-point connections with

a Commercial-Off-The-Shelf (COTS) network technology (e.g. Ethernet, Infiniband, Omnipath). FELIX servers will act as a routers between 24-48 detector serial links and 2-4 standard 40Gbps/100Gbps links. FELIX servers will communicate with a smaller set of commercial servers, known as Software ReadOut Drivers (SW ROD), used for data collection and processing of physics data. In addition, different components need to connect to the FELIX servers. For example, the Detector Control System (DCS) monitors and controls the detector front-end electronics while the Control & Configuration system sets up and manages data acquisition applications.

The FELIX project is planned to be implemented in two phases. In 2018-2019 some detector hardware will be moved to this new schema (approx. 68 FELIX and 44 SW ROD servers will be installed). A complete migration of the remaining hardware is planned for 2025. Part of this effort consists of designing and implementing a network that can meet the demands of the system (high-availability, high-throughput, low-latency, redundancy, etc.)

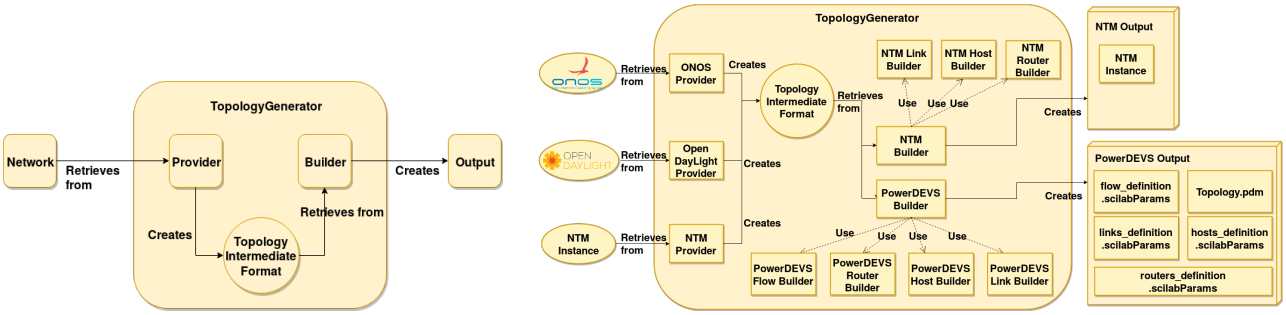
Dataflow modeling and simulation methodology supports the design of the network and aids in the decision process (e.g. for selecting technologies, topologies, node distributions, etc.). Yet, the generation of many possible simulation scenarios to be evaluated is currently a manual process which is time-consuming, error-prone and does not provide an automated update procedure.

5 TOPOGEN: A TOPOLOGY GENERATION AND TRANSFORMATION ARCHITECTURE

TopoGen is a flexible architecture for network topology description, generation and translation. Yet, its conception was motivated by technology-specific needs, namely, to obtain a graph model representation of a network topology by querying SDN controllers (such as ONOS and OpenDayLight) and to generate DEVS simulation models for the PowerDEVS tool (according to the goals described in section 4). In this section we describe the overall architecture of TopoGen and some illustrative implementation-related details using the ONOS SDN controller as a sample source for topology information, and PowerDEVS as a sample destination for simulation.

5.1 Architecture

In Figure 2 (a) the architectural module viewtype of TopoGen is outlined. Its main components are:



(a) TopoGen high-level architectural view

(b) Providers and Builders currently implemented

Figure 2: TopoGen architectural views.

- **Network**: A network description to be loaded, modified or translated. It can be either a real network (e.g. one described by an SDN controller) or a virtual one described by some description language.
- **Provider**: A component that handles the interaction with the *Network* component. A *Provider* retrieves (parses) network information (such as nodes, resources, connections) and translates it into a common *Topology Intermediate Format*. A *Provider* component is specialized to the *Network* component to be accessed. Built-in providers are discussed in section 5.2.

- **Topology Intermediate Format (TIF)**: The internal in-memory representation of a network topology, written by any *Provider* and read by any *Builder*. The main goal of a *TIF* is to serve as an internal abstraction layer that permits orchestration of different *Builders* and *Providers* in a flexible way.
- **Builder**: A component that parses a TIF and serializes it according to a desired output. A *Builder* component is specialized to the *Output* component to be generated. Built-in builders are discussed in section 5.2.
- **Output**: An output format that some *Builder* must comply with in order to perform a translation from the TIF format. The *Output* can consist of a single file or a set of files, depending on the requirements of the software tool that will ultimately consume them.

5.2 Built-in Providers and Builders

Figure 2 (b) shows a more detailed component diagram of TopoGen as developed for this work. It shows different *Builder* implementations according to the requirements of the expected *Output*. The implemented components are:

- **Providers. Network Topology Model (NTM)**: The NTM provider serializes topologies described with NTM, an object-oriented network topology description language (NTM is based on Ruby, and is detailed in section 5.3). NTM can be used to model existing networks or to draft new designs. **ONOS and OpenDayLight**: Providers for the ONOS and OpenDayLight SDN Controllers retrieve topology information by accessing their exposed APIs. These are examples of parsing existing operative networks.
- **Builders. NTM**: This builder serializes a network described in the TIF format, creating a new NTM instance. This instance is typically used to programmatically customize a topology retrieved from different sources before generating a final target *Output*. An example of topology augmentation is presented in section 6. **PowerDEVS**: This builder creates a PowerDEVS model simulation structure. It relies on parameterizable DEVS atomic models that provide basic behavioural building blocks. The builder hierarchically composes DEVS atomic models to create DEVS coupled models with more complex behaviours. Typical DEVS atomic models are queues, links, etc. For instance, to compose a switch, several atomic models of input/output queues are composed together and parameterized as Prioritized Queues with the Quality of Service (QoS) flag activated. Meanwhile, in order to compose a regular host, only one input/output queue is needed parameterized as a standard non-prioritized NIC queue.

Providers and *Builders* are decoupled by means of the Topology Intermediate Format. Any provider implementation can be used with any builder implementation. TopoGen can be extended by defining new providers and builders at will. Provider and Builder classes implement a Strategy Pattern, where each strategy is implemented outside TopoGen. This allows TopoGen to be adapted to different scenarios while implementing each provider and builder only once.

5.2.1 Class Diagram

Figure 3 presents a class diagram for the TopoGen implementation in the Ruby language.

When TopoGen is used, an instance of *TopologyGenerator* class is created and the initialize method is invoked with parameters denoting a provider, a builder (the directory where the output will be stored) and a URI (from where to retrieve the topology from). The *TopologyGenerator* class has one *TopologyProvider* and one *OutputBuilder* instance. The *TopologyProvider* class can be mapped to the *Provider* component showed in Figure 2 (a). This class has four children: *OnosTopologyProvider* and *OpenDaylightProvider* classes encapsulate the logic for retrieving information from SDN controllers' APIs. The *ObjectTopologyProvider* class retrieves a topology from a *Topology* instance. Finally, the *CustomTopologyProvider* class retrieves information from a NTM instance.

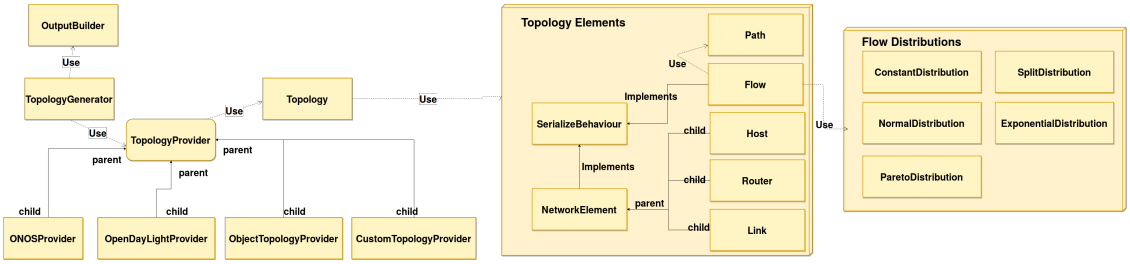


Figure 3: TopoGen class diagram.

The *Topology* class in Figure 3 plays the role of the TIF in the architecture (Figure 2 (a)). It uses TopologyElements classes to represent the elements in the network. A *Topology* can have multiple TopologyElements, however every TopologyElement belongs to a unique *Topology*. The TopologyElements box contains all the classes that can be used to create elements in a *Topology* instance. The *NetworkElements* class represents an abstraction of the physical elements of the network (in this case *Host*, *Link* and *Router*). The *Flow* and *NetworkElement* classes implement a *SerializeBehaviour*, which is a module for serializing classes. The *Flow* class represents a flow of packets between hosts. When creating a *Flow* instance, a packet rate distribution and a packet size distribution are needed. Distribution classes are shown in the Flow Distribution box. Finally, the *OutputBuilder* class represents the component *Builder* in Figure 2 (a).

5.3 The Network Topology Model

The Network Topology Model (NTM) is an object oriented approach to represent data networks in Ruby. NTM makes it possible to describe all the elements in a network: physical elements (e.g. hosts, routers, links, etc.), and logical elements (e.g. data flows, routing paths, etc). NTM is currently dependent on TopoGen as it was designed to be used by the CustomTopologyProvider class (see Figure 3).

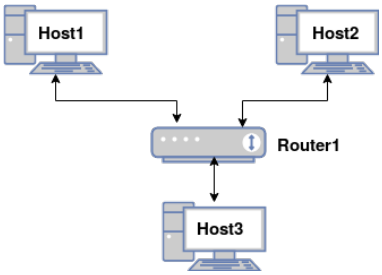


Figure 4: Simple network topology to be described with NTM.

The following NTM Ruby code describes the network shown in Figure 4 including the communication flow between Host1 and Host3:

```
1  module NetworkTopology
2   def get_topology
3    return @topology.topology_elements unless @topology.topology_elements.size == 0
4    hosts = []
5    router = @topology.add_router "Router1"
6    for i in 0..2
7      host = @topology.add_host "Host#{i}"
8      hosts.push host
9    end
10   bwidth = 500*1000*1000 # 500 Mbps
11   @topology.add_full_duplex_link "Link1",hosts[0],0,router,0,bwidth
12   @topology.add_full_duplex_link "Link2",hosts[1],0,router,1,bwidth
```

```
13   @topology.add_full_duplex_link "Link3",hosts[2],0,router,2,bwidth
14   link1 = @topology.get_element_by_id "Link1_up"
15   link2 = @topology.get_element_by_id "Link3_down"
16   flow_1_path = Path.new hosts[0], hosts[2]
17   flow_1_path.add_link link1
18   flow_1_path.add_link link2
19   @topology.add_flow "Flow1",10,[flow_1_path],(ExpDistrib.new 1.0/6875),(
         ConstDistrib.new 1000*8)
20   @topology.topology_elements
21  end
22 end
```

Each NTM instance must define a *NetworkTopology* module (line 1) and a *get_topology* method (line 2) which returns the elements added in the Topology instance (variable *@topology*). To create the topology, the router is first added in line 5 (*add_router* method). The hosts are defined in lines 6-9 (*add_host* method) with a unique identifier for each host. In lines 11-13 links are added using *add_full_duplex_link*. This method expects an ID, a source element (an instance of Router or Host), a source port number, a destination element, a destination port number and the bandwidth (in bps), and creates two source/destination links.

The first link goes from source to destination (its ID is the concatenation of strings *up* and the ID receives). The second link goes from destination to source (its ID is the concatenation of strings *down* and the ID received). The method returns the second link only. In lines 14 and 15 the links that define a path between Host1 and Host3 are retrieved. In lines 16 to 18 a new path is created between Host1 and Host3 using the retrieved links. In lines 19 to 20 a new flow is added with the *add_flow* method (it expects an ID, a flow priority, an array of possible paths for the flow, and stochastic distributions for packet rate and size). The NTM description ends by returning the new elements with the *topology_elements* method.

6 SUPPORT FOR DESIGNING THE FELIX NETWORK AT THE ATLAS DATA ACQUISITION SYSTEM

In this section we describe TopoGen as applied in a real world scenario. The case study builds upon a modeling and simulation-driven engineering process (Bonaventura et al. 2016) developed for the ATLAS TDAQ network at CERN (Pozo Astigarraga et al. 2015). We show how TopoGen can assist the design phase for the network to be implemented 2019-2020 in the ATLAS FELIX project (Anderson et al. 2015).

6.1 The FELIX Network Requirements

The FELIX network will provide connectivity between different components of the FELIX system (see Figure 1) and will handle various types of traffic which differ in their throughput, latency, priority and availability requirements. For example, the Detector Control System (DCS) that monitors and controls the detector's front-end electronics requires the highest priority and low latency to react fast, but is expected to require low throughput. Meanwhile, the detector's data will use most of the network bandwidth so it can have less priority to avoid saturation. Table 1 summarizes the different traffic types and their requirements.

The communication patterns are also different for each type of traffic. While DCS traffic follows a many-to-one pattern (all FELIX servers communicate with a single DCS server), Control and Monitoring traffic require a many-to-few pattern. Detector data, on the other hand, uses a simple one-to-one or two-to-one pattern from a FELIX server to SW RODs.

To provide confidence about the coexistence of these traffic types while meeting performance requirements we adopt a modeling and simulation approach to study expected throughput and latency for each traffic type, and anticipate possible bottlenecks. Although the high level requirements are defined, each subsystem's specification is updated often during the design process. Specific subsystem parameters (throughput, processing times, etc.) will not be known until the final system is in place. Yet, simulation can provide guidelines for realistic ranges of candidate parameter values (parameter sweeping).

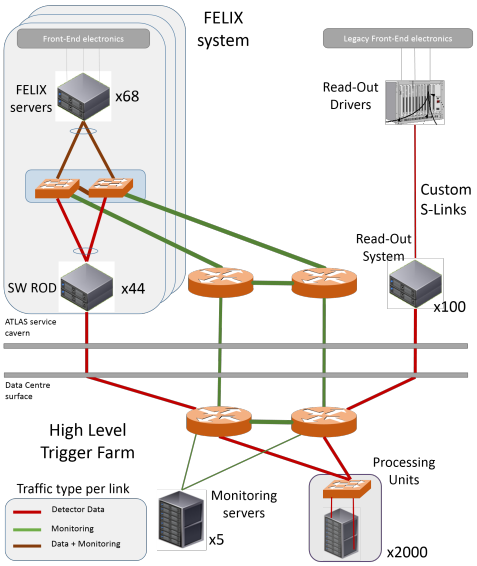


Figure 5: Topology of the FELIX system.

Table 1: FELIX traffic types and requirements.

Traffic type	Throughput	Latency	Priority	Comm. Pattern
DCS	Low	Low	High	Many to one
Control and Config.	Med.	-	High	Many to few
Detector Data	High	-	Med.	One to one
Monitoring	High	-	Low	Many to few

Technologies rely on different protocols, congestion control algorithms, and routing schemes which also need to be considered in simulation studies. In particular, different restrictions are imposed over candidate topologies depending on selected technologies (e.g. Ethernet allows for heterogenous link speeds, Infiniband does not. Infiniband efficiently supports mesh and leaf-spine topologies, Ethernet supports topologies with cycles but using algorithms that perform poorly). The simulation platform needs to be able to define all these types of topologies in a flexible way to support agile design iterations.

6.2 Using TopoGen to Support the Modeling and Simulation Process

Figure 6 shows the workflow used while applying TopoGen to aid the modeling and simulation process for the FELIX network. The workflow consists of three phases: first, the topology under design by the networking team is automatically retrieved and serialized into an NTM instance; second, the NTM topology is augmented programmatically with extra resources (nodes and data flows), and third the new topology is serialized into a PowerDEVS simulation model. Hence a simulation model is automatically created from an existing specification originally meant for other purposes. This workflow deals with topology changes at design-time. Run-time adaptation of simulations to topology changes remains a subject of future work.

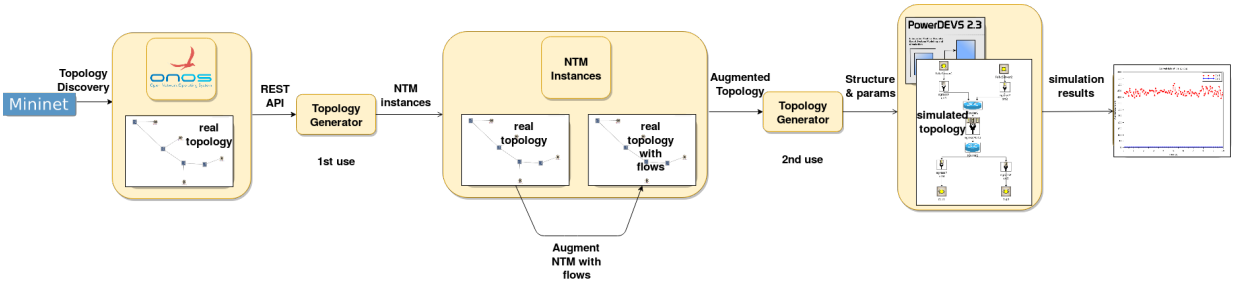
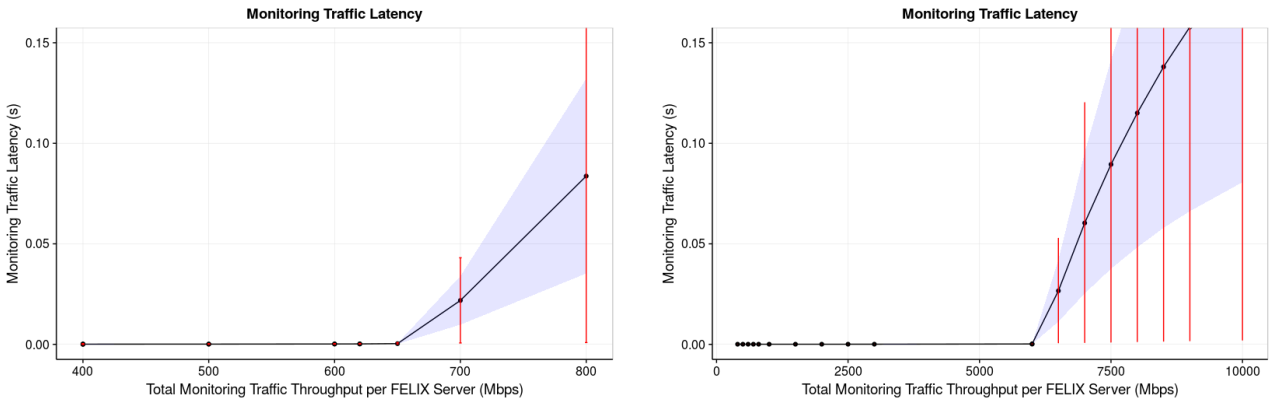


Figure 6: Modeling and simulation workflow using TopoGen.

In the **first phase**, the networking team provided a Mininet emulated environment used to test the connectivity of a topology, including nodes only from the FELIX network. The ONOS SDN controller was installed within the emulated environment to provide network discovery services. Then, the TopoGen ONOS Provider was configured to connect with the REST API exposed by ONOS to query the topology.



(a) (b)

Figure 7: Simulated mean packet latency seen by the Traffic Monitoring servers. Link capacity allocated to monitoring traffic: (a) 1 Gbps (b) 10 Gbps. Blue area: standard deviation. Red lines: min-max range.

Once the topology is retrieved, the TopoGen NTM Builder serialized it into an NTM instance for later use. Each time the networking team updates their emulated topology, it can be retrieved again to keep the NTM and simulation models up-to-date. The fact that the original topology was specified in an emulated environment is transparent for TopoGen. In the **second phase**, additional nodes are added to the NTM topology to also represent the HLT network (see Figure 5). To generate a meaningful simulation model extra information is needed about the traffic generated by different servers. Nodes and data flows, along their respective parameters, were added programmatically into the original NTM instance guided by the network engineers. For this case study, only the Detector Data traffic type and the Monitoring traffic type were considered (see Table 1). In the **third phase**, the augmented NTM instance is loaded by the TopoGen NTM Provider and used by the PowerDEVS Builder to generate all necessary files for simulation.

The network actually simulated with PowerDEVS is the one presented in Figure 5. It includes the FELIX network nodes (automatically retrieved from the SDN controller) and the HLT network nodes, the Detector Data and Monitoring traffic flows (added programmatically with NTM). This case study focused on network behaviour under different intensities of Monitoring traffic rates, matching an engineering requirement.

6.3 Simulation Results

We studied the potential effects on the average latency of FELIX Monitoring traffic in the case of an upgrade of the bottleneck links from 1 Gbps to 10 Gbps. Figure 7 (a) shows the average packet latency for FELIX Monitoring flows in different scenarios with increasing monitoring throughput for all servers. As monitoring traffic grows the latency slightly increases until a transition is observed at the point when each server generates 650 Mbps of monitoring data. After that point the latency increases rapidly denoting the presence of congestion. The buffer sizes and link utilization at the switches (not included in this report) indicate that the source of congestion are the 1 Gbps links of monitoring servers. We then updated the topology in the NTM instance, now with a link capacity of 10 Gbps for monitoring nodes. The PowerDEVS simulation model was regenerated with TopoGen, and new experiments were run. Figure 7 (b) shows how the saturation point moves up to 6500 Mbps of monitoring traffic. The congestion point in the topology remains at the links directly connecting the monitoring servers.

7 CONCLUSIONS

We presented TopoGen, a reference architecture and tool for systematic translation and generation of network topologies. We also introduced NTM, a network topology model to describe networks programmatically

using the Ruby language. Decoupled Providers, Builders, and a Topology Intermediate Format allow the creation of flexible topology transformation workflows that can suit diverse needs. We focused on the generation of DEVS simulation models starting from network information available at SDN Controllers.

We applied TopoGen successfully in the context of a real-world network design process: the FELIX system of the ATLAS TDAQ network at CERN. TopoGen proved effective to retrieve a large topology from an ONOS controller, export it to a programmable network model, augment the model manually according to particular simulation needs, and generate a fully operational DEVS simulation for the PowerDEVS tool.

When compared with previous experiences in our team achieving the same results and in the same context, TopoGen strongly reduced the time to completion, complexity and error-proneness. Future steps for TopoGen include increasing the number of available Providers to include more SDN controllers and network description formats, as well as new Builders for other DEVS and also non-DEVS based simulators.

REFERENCES

Anderson, J., A. Borga, H. Boterenbrood, H. Chen, K. Chen, G. Drake, D. Francis, B. Gorini, F. Lanni, G. L. Miotto et al. 2015. "FELIX: A High-Throughput Network Approach for Interfacing to Front end Electronics for ATLAS Upgrades". In *Journal of Physics: Conf. Series*, Volume 664, 082050. IOP.

ATLAS Collaboration 2008. "The ATLAS Experiment at the CERN Large Hadron Collider". *Journal of Instrumentation* 3 (08): S08003.

Berde, P., M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow et al. 2014. "ONOS: Towards an Open, Distributed SDN OS". In *Proceedings of the third workshop on Hot topics in software defined networking*, 1–6. ACM.

Bergero, F., and E. Kofman. 2011. "PowerDEVS: a Tool for Hybrid System Modeling and Real-Time Simulation". *Simulation* 87 (1-2): 113–132.

Bergero, F., and E. Kofman. 2014. "A Vectorial DEVS Extension for Large Scale System Modeling and Parallel Simulation". *Simulation* 90 (5): 522–546.

Bjorklund, M. 2010, October. "YANG-A Data Modeling Language for the Network Configuration Protocol". RFC 6020, IETF.

Bonaventura, M., D. Foguelman, and R. Castro. 2016. "Discrete Event Modeling and Simulation-Driven Engineering for the ATLAS Data Acquisition Network". *Computing in Science & Engineering* 18 (3): 70–83.

Bonaventura, M., G. Wainer, and R. Castro. 2013. "Graphical Modeling and Simulation of Discrete-Event Systems With CD++ Builder". *Simulation* 89 (1): 4–27.

Carneiro, G. 2010. "NS-3: Network Simulator 3". In *UTM Lab Meeting April*, Volume 20.

Castro, R., and E. Kofman. 2015. "An Integrative Approach for Hybrid Modeling, simulation and control of data networks based on the DEVS formalism". In *Modeling and Simulation of Computer Networks and Systems: Methodologies and Applications*, Chapter 18. Morgan Kaufmann.

Chang, X. 1999. "Network simulations with OPNET". In *Proceedings of the 1999 Winter Simulation Conference*, edited by P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, 307–314. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

De Oliveira, R. L. S., A. A. Shinoda, C. M. Schweitzer, and L. R. Prete. 2014. "Using mininet for emulation and prototyping software-defined networks". In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, 1–6. IEEE.

Fernandes, S. 2017. *Performance Evaluation for Network Services, Systems and Protocols*. Springer.

Foguelman, D., M. Bonaventura, and R. Castro. 2016. "MASADA: A Modeling and Simulation Automated Data Analysis framework for Continuous Data-Intensive validation of Simulation Models". In *30th Annual European Simulation and Modelling Conf.*, Volume 30, 34–42. Eurosis.

Großmann, M., and S. J. Schuberth. 2013. "Auto-Mininet: Assessing the Internet Topology Zoo in a Software-Defined Network Emulator". *Messung, Mellierung un Bewertung von Rechensystemen (MMBnet)* 7.

Himsolt, M. 2010. "GML: A Portable Graph File Format". Technical report, Universität Passau.

Knight, S., H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. 2011. "The internet Topology zoo". *IEEE Journal on Selected Areas in Communications* 29 (9): 1765–1775.

Koslovski, G. P., P. V.-B. Primet, and A. S. Charao. 2008. "VXDL: Virtual resources and Interconnection Networks Description Language". In *Intl. Conf. on Networks for Grid Applications*, 138–154. Springer.

Kreutz, D., F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. 2015. "Software-defined Networking: A Comprehensive Survey". *Proceedings of the IEEE* 103 (1): 14–76.

Medved, J., R. Varga, A. Tkacik, and K. Gray. 2014. "Opendaylight: Towards a Model-Driven SDN Controller Architecture". In *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*, 1–6. IEEE.

Ngangue Ndih, E. D., and S. Cherkaoui. 2015. "Simulation Methods, Techniques and Tools of Computer Systems and Networks". In *Modeling and Simulation of Computer Networks and Systems: Methodologies and Applications*, edited by M. S. Obaidat, F. Zarai, and P. Nicopolitidis, Chapter 17. Morgan Kaufmann.

Pozo Astigarraga, M., E. ATLAS Collaboration et al. 2015. "Evolution of the ATLAS Trigger and Data Acquisition System". In *Journal of Physics: Conf. Series*, Volume 608, 012006. IOP.

Quesnel, G., R. Duboz, and É. Ramat. 2009. "The Virtual Laboratory Environment–An Operational Framework for Multi-Modelling, Simulation and Analysis of Complex Dynamical Systems". *Simulation Modelling Practice and Theory* 17 (4): 641–653.

Van der Ham, J., P. Grosso, R. Van der Pol, A. Toonk, and C. De Laat. 2007. "Using the Network Description Language in Optical Networks". In *Integrated Network Management, 2007. im'07. 10th IFIP/IEEE International Symposium on*, 199–205. IEEE.

Van Tendeloo, Y., and H. Vangheluwe. 2017. "An Evaluation of DEVS Simulation Tools". *Simulation* 93 (2): 103–121.

Varga, A., and R. Hornig. 2008. "An Overview of the OMNeT++ Simulation Environment". In *Proceedings of the 1st international conf. on Simulation tools and techniques for communications, networks and systems*, 60. ICST.

Wehrle, K., M. Günes, and J. Gross. 2010. *Modeling and Tools for Network Simulation*. Springer.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic press.

Zhao, Y., L. Iannone, and M. Riguidel. 2015. "On the Performance of SDN Controllers: A Reality Check". In *IEEE Conf. on Network Function Virtualization and Software Defined Networks*, 79–85. IEEE.

AUTHOR BIOGRAPHIES

ANDRÉS LAURITO is an MASc student in Computer Science in the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. His thesis explores new intent programming languages for SDN with semantic checking. His email address is alaurito@dc.uba.ar

MATAS BONAVENTURA is an MASc in Computer Science and a PhD student in the Departamento de Computacin, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires and a project associate with the ATLAS TDAQ group (CERN). His research interests are hybrid continuous/discrete modeling and simulation of networked computing systems. His email address is mbonaventura@dc.uba.ar.

MIKEL EUKENI POZO ASTIGARRAGA is a data acquisition engineer in the ATLAS TDAQ group (CERN). He specializes in high throughput-high availability networks, with research interests include SDN and data center protocols. His email address is mikel.eukeni.pozo.astigarraga@cern.ch.

RODRIGO CASTRO is a Professor in the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, head of the Simulation Lab, and a researcher at CONICET. His research interests include simulation and control of hybrid systems. His email address is rcastro@dc.uba.ar.