



Article

Experimenting with Routing Protocols in the Data Center: An ns-3 Simulation Approach

Leonardo Alberro ^{*}, Felipe Velázquez, Sara Azpiroz, Eduardo Grampín  and Matías Richart 

Instituto de Computación (InCo), Universidad de la República (UdelaR), Montevideo 11300, Uruguay

^{*} Correspondence: lalberro@fing.edu.uy

Abstract: Massive scale data centers (MSDC) have become a key component of current content-centric Internet architecture. With scales of up to hundreds of thousands servers, conveying traffic inside these infrastructures requires much greater connectivity resources than traditional broadband Internet transit networks. MSDCs use Fat-Tree type topologies, which ensure multipath connectivity and constant bisection bandwidth between servers. To properly use the potential advantages of these topologies, specific routing protocols are needed, with multipath support and low control messaging load. These infrastructures are enormously expensive, and therefore it is not possible to use them to experiment with new protocols; that is why scalable and realistic emulation/simulation environments are needed. Based on previous experiences, in this paper we present extensions to the ns-3 network simulator that allow executing the Free Range Routing (FRR) protocol suite, which support some of the specific MSDC routing protocols. Focused on the Border Gateway Protocol (BGP), we run a comprehensive set of control plane experiments over Fat-Tree topologies, achieving competitive scalability running on a single-host environment, which demonstrates that the modified ns-3 simulator can be effectively used for experimenting in the MSDC. Moreover, the validation was complemented with a theoretical analysis of BGP behavior over selected scenarios. The whole project is available to the community and fully reproducible.



Citation: Alberro, L.; Velázquez, F.; Azpiroz, S.; Grampín, E.; Richart, M. Experimenting with Routing Protocols in the Data Center: An ns-3 Simulation Approach. *Future Internet* **2022**, *14*, 292. <https://doi.org/10.3390/fi14100292>

Academic Editor: Michael Sheng

Received: 08 September 2022

Accepted: 10 October 2022

Published: 14 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: ns-3; routing; data center

1. Introduction

Content-centric, cloud-based networking is the dominant model in the current Internet, with the pervasive presence of content providers with data center infrastructures deployed throughout the whole world. Content delivery networks (CDNs) replicate content in locations close to users, in order to improve their quality of experience, while over the top (OTT) providers behave in a similar way, consolidating the Internet distributed data center model. Moreover, resource virtualization is making its contribution to the Internet architecture shift as well, since the usual way to deploy online applications is to use cloud computing providers, which, not surprisingly, also base their operations on data centers with ubiquitous connectivity.

Both content and computing business are based in huge data centers with similar basic functions such as compute, store, and replicate data using message exchange among servers, taking advantage of supporting communication infrastructure. These data centers, which may comprise hundreds of thousands of servers, are called massive scale data centers (MSDC).

The traffic between users and applications running in the data center is called north-south traffic, while on the other hand, east-west traffic is the one exchanged by servers within the data center; the latter represents 85% of the total [1].

The traffic demand in MSDC, much higher than the traditional internet, requires specific solutions at the forwarding, routing and transport levels, taking advantage of the topological possibilities offered by Fat-Trees, inspired by Clos networks [2]. These

networks, originally conceived to build non-blocking switching matrices for telephone networks, are made up of multiple levels of switches, where each switch of one level is connected to all those of the next level, obtaining a high path diversity as a result.

In a previous work [3], we experimented with data center routing protocols in emulated environments such as Kathara [4,5], Megalos [6], CORE [7] or Mininet [8], complementing the work presented in [9], where the Sibyl framework is used for evaluating implementation of routing protocols in fat-trees, including the Border Gateway Protocol (BGP) in the data center [10], Openfabric (IS-IS with flooding reduction) [11], and Routing in Fat Trees (RIFT) [12,13]. This framework presents wall-clock independent metrics, which permits us to normalize the results disregarding the underlying execution environment.

These previous works are based on the routing protocols from the Free Range Routing (FRR) suite, an open source implementation of BGP, OSPF, RIP, IS-IS, and other protocols, inheriting the code base of the Quagga project [14].

As mentioned in the previous work, emulated devices run exactly the same firmware of hardware devices, therefore implementing identical functionality. Moreover, emulated devices are exposed to real-life software errors, which permits us to not only evaluate functionality, but also resilience. On the other hand, re-implementation of network protocols and applications is needed for discrete event simulation, weakening the chances of testing real use cases. Nevertheless, a simulator provides an environment for replicable experiments always guaranteeing the same conditions and provides fine management of the timing issues.

With these considerations in mind, in this paper we present a port of FRR, to the Direct Code Execution (DCE) [15] mode of the ns-3 Network Simulator [16]. ns-3 is a discrete event network simulator for Internet systems, widely supported in the networking community. ns-3 has a mode of execution called DCE [17], which allows using native code (properly compiled) in the simulations. In this way, it is possible to execute existing implementations of network protocols or applications within ns-3. Therefore, it is possible to reconcile the virtues of discrete event simulators with emulation, which preserves the real implementation of protocols and applications. Moreover, this approach permits us to run a fair comparison among different experimentation frameworks which run FRR.

Thus, we seek to perform the necessary implementations so that ns-3 can support FRR, in order to develop simulations in ns-3 DCE that use FRR code. While FRR implements a set of network protocols, the scope of this work is to support the implementation provided for BGP. For the implementation of the simulations, we will focus on the fat tree CLOS topology, which is widely used in massive data centers. The aim is to study the behavior of the BGP protocol in this context.

The main contribution of our work consists of a simulation platform to test and analyze routing protocols in the context of MSDCs. This platform provides the ability to simulate the FRR suite and in particular the MSDC routing algorithms. To achieve this, our work includes: (i) an extension of DCE to support the FRR suite, (ii) a *FRRHelper* class, which facilitates the instantiation and usage of FRR in a simulation script, (iii) an extension of the fat-tree topology generator *VFTGen* [18] to produce ns-3 simulation scripts, (iv) a comparison and validation between emulation and simulation-based approaches for BGP in data center, and (v) a comparison and validation between the experimental results and a theoretical analysis of BGP behavior over selected scenarios.

The remainder of this paper is organized as follows: Section 2 provides the background and presents the Sibyl framework as related work. Section 3 describes the process of porting FRR to ns-3, using the DCE module. Section 4 exposes the validation of the port and the experimental results generated. Consequently, a basic functional evaluation is described. Secondly, a comparison against the Sibyl framework is carried out. Finally, a validation against a theoretical analysis of BGP behavior over two selected scenarios is performed. In Section 5, a performance analysis is exposed. It evaluates the scalability, memory usage and execution times for different network sizes. Additionally, two features to improve the performance of the port are described; and a comparison between the execution times in

the simulated and emulated environment is exhibited. Finally, in Section 6, we discuss the most relevant aspects and conclusions of this work.

2. Background and Related Work

There are different approaches to network control plane debugging, namely model-based verification, and testing over emulation or simulation environments. In this work, we concentrate on testing tools. Regarding emulation tools, we have been working with scalable environments such as CORE, Mininet, Kathará and Megalos, where an actual protocol implementation can be tested in a controlled environment. Moreover, the Sibyl framework, which works over Kathará and Megalos, assembles different tools for protocol evaluation over fat-trees.

In the case of simulations, re-implementation is often needed. This presents a major drawback for protocol debugging, and therefore it is not the most usual path to follow. Some previous works have attempted to offer real code execution over a simulator but, to the best of our knowledge, only DCE has a working environment tested with many real world implementations. In Section 3, we present in more detail the characteristics of ns-3 and DCE, and the FRR port effort.

2.1. The Sibyl Framework

In this section, we will briefly describe the Sibyl framework that we will use as a baseline for comparison and validation of our proposal, given the public availability of a complete data-set of experiments [19]

Kathará is a network emulation system that accurately reproduces the behavior of a real system, using Docker containers [20] to implement devices, which represents a lightweight alternative to standard virtualization solutions, allowing devices to use different images in the same network scenario (for example, different implementations of a given network protocol).

Kathará supports different virtualization managers, and in order to support horizontal scalability, it uses Kubernetes [21], adopting the name Megalos. Since it runs distributed in a cluster of servers, the low level connectivity of emulated devices is implemented using a Virtual Extensible LAN (VXLAN) data plane with an EVPN BGP control plane.

The Sibyl framework integrates the aforementioned environments, tailored to perform a large number of experiments on parametric fat-tree topology configurations. During each experiment, Sibyl performs a series of steps, starting by generating a topology, deploy nodes running specific containers and network links, start the experiment and capture relevant PDUs, shutting down and analyzing the results (for further details, see [9]).

We used the results gathered following these steps as a baseline for comparison with other experimentation environments, in particular with the FRR port to ns-3 presented in this paper.

2.1.1. Sibyl Fat-Tree Experimentation Tools

In this section, we describe the tools included in the Sibyl framework, as follows:

- VFTGen [18] automatically generates and configures fat-tree topologies for Sibyl. It takes as input the parameters of a fat-tree.
- Sibyl RT Calculator is a tool for generating the expected forwarding tables of the network nodes of a fat-tree, taking into account the routing protocol (e.g., BGP) and the type of test (e.g., Node Failure).
- Sibyl Analyzer is a tool to analyze the results of the experiments using the packets exchanged by the nodes during an experiment.

2.1.2. The Timing Issue

Sibyl implements a wall-clock independent metric, which permits us to normalize the results disregarding the underlying execution environment.

This is necessary for emulated environments, where underlying hardware resources cannot be taken for granted. On the other hand, execution time is completely under control in discrete event simulations, permitting us to measure performance parameters with certainty. This is the main reason to attempt the FRR port to ns-3, along with the fact that DCE permits us to execute native code.

2.2. Fat Tree Networks

Fat-tree networks are topologically partially ordered graphs, and “level” denotes the set of nodes at the same height in such a network, where the nodes of level zero (the lowest) are called Leaves, those of level one are Spines, and the ones of level two are Top of Fabric (ToF) or Cores. The subset of Leaf and Spine nodes that are fully interconnected is called a Point of Delivery (PoD). Level two is called the aggregation level and has the responsibility of connecting different PoDs.

Following the notation described in [12], a fat-tree topology can be specified by three parameters: K_{LEAF} , K_{TOP} and R . K_{LEAF} and K_{TOP} describe the number of ports pointing north or south for the leaf and spine nodes, respectively. Finally, the number of links from a ToF to a PoD are denoted by R and called “redundancy factor”. As an example, the Figure 1 shows a fat-tree with $K_{LEAF} = 2$, $K_{TOP} = 2$ and $R = 1$. For simplicity, from now on we assume $K_{LEAF} = K_{TOP} = K$.

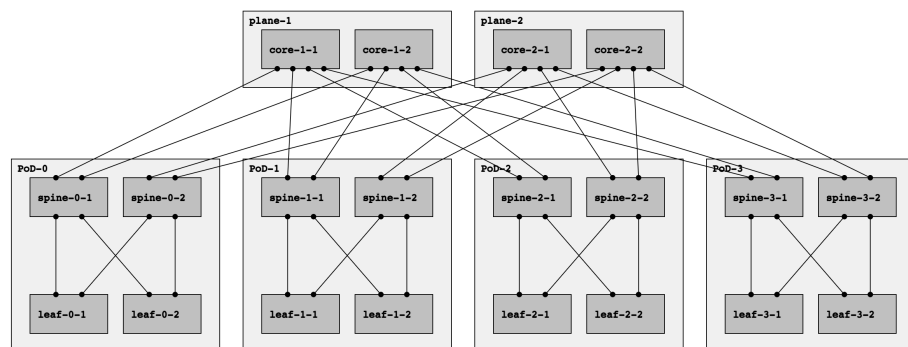


Figure 1. A fat-tree multi-plane topology with $K = 2$, $R = 1$ and $N = 2$ planes.

Observe that there are two types of fat trees: single-plane and multi-plane. In a single-plane topology, each ToF is connected to all the Top of PoD (ToP). This topology has the maximum value of redundancy factor, with $R = K$. In these topologies, the number of ports for each ToF is at least $P \times K$, which might be unfeasible if P and/or K are too large.

On the other hand, in a multi-plane topology, ToFs are partitioned into planes: $N = K/R$ sets, each with the same number of nodes. All the ToFs of the same plane are connected to the same set of spines of each PoD. The topology shown in the Figure 1 can be described as a multi-plane fat-tree with $K = 2$ and $R = 1$ and $N = K/R = 2$ ToF planes. It is worth noting that in this configuration, redundancy is sacrificed to increase the number of PoDs.

3. FRR Port to ns-3 DCE

In this section, we detail the process of porting FRR to ns-3, using the DCE module. The process involved: (i) changes to DCE to be able to execute the FRR code, which implied re-implementing some functions from the C library (glibc) that are used by FRR, also fixing some *bugs* found in existing DCE code; (ii) minor changes to the code of FRR, in order to solve some problems that were difficult to find another solution to; (iii) implementing a class `FrrHelper` in a way that makes it easy to write scripts that use the port, and (iv) carrying out tests in order to evaluate and validate the port, which we will see in Sections 4 and 5. The aforementioned port is open source and is available at [22].

3.1. Background on ns-3 and DCE

ns-3 [16] is a discrete-event network simulator used mainly in research and education. It is open-source and free, licensed under the GNU GPLv2 license.

Both ns-3 core and models are implemented in C++. It is built as a library that can be linked both statically and dynamically by a main C++ program, which defines the network topology and starts the simulation [23]. Typically, to run a simulation in ns-3, a C++ program is created (script in the ns-3 nomenclature) that defines the topology and configuration for the simulation. This program includes at the end a call to the `Run()` function of the `Simulator` class that will start the simulation.

Regarding Direct Code Execution (DCE) [15], it is a framework for ns-3 that allows us to execute existing implementations of network applications or protocols within ns-3 without any changes to the source code. This permits us to execute existing real applications such as the ping application or even more, the entire Linux networking stack within an ns-3 simulation.

Thus, in a ns-3 simulation which uses DCE, the network topology as well as channel configurations will be done in ns-3, while applications running on nodes can use DCE, including Linux native applications or actual implementations of network protocols, such as Linux's TCP, as shown in Figure 2.

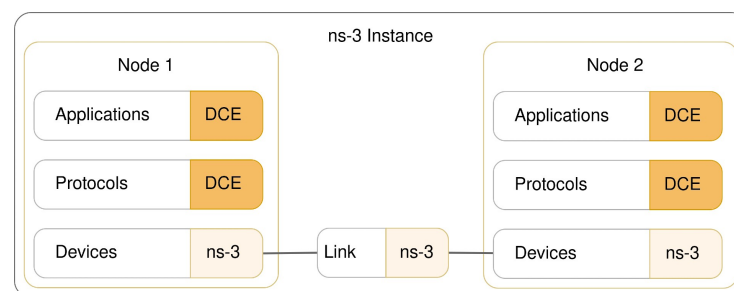


Figure 2. DCE is used for running Linux applications without code changes. On top of that, it enables the use of the Linux network protocol stack in ns-3 simulations. Net devices (and channel) are simulated only with ns-3, while applications and network protocols can use DCE.

There are two ways to run DCE: basic mode and advanced mode. Basic mode uses the ns-3 networking stack, while advanced mode uses the Linux networking stack. The latter is done using the Linux kernel as a library.

The design of DCE takes its idea from the library operating system (*LibOS* [24]). DCE is structured around three components: Core, Kernel and POSIX, as shown in Figure 3. First, at the bottom level is the Core module that handles memory virtualization: stack, heap and global variables. Above that is the Kernel layer that takes advantage of these services to provide an execution environment for the Linux network stack within the simulator. For Advanced Mode, DCE uses the Linux kernel implementation of layer 3 and 4 protocols and Layers 1 and 2 are simulated with ns-3. DCE takes care of synchronization, making the Linux kernel see ns-3 network devices as if they were real devices. Finally, the POSIX layer builds on top of the Core and Kernel layers to re-implement the standard socket API for use by simulated applications.

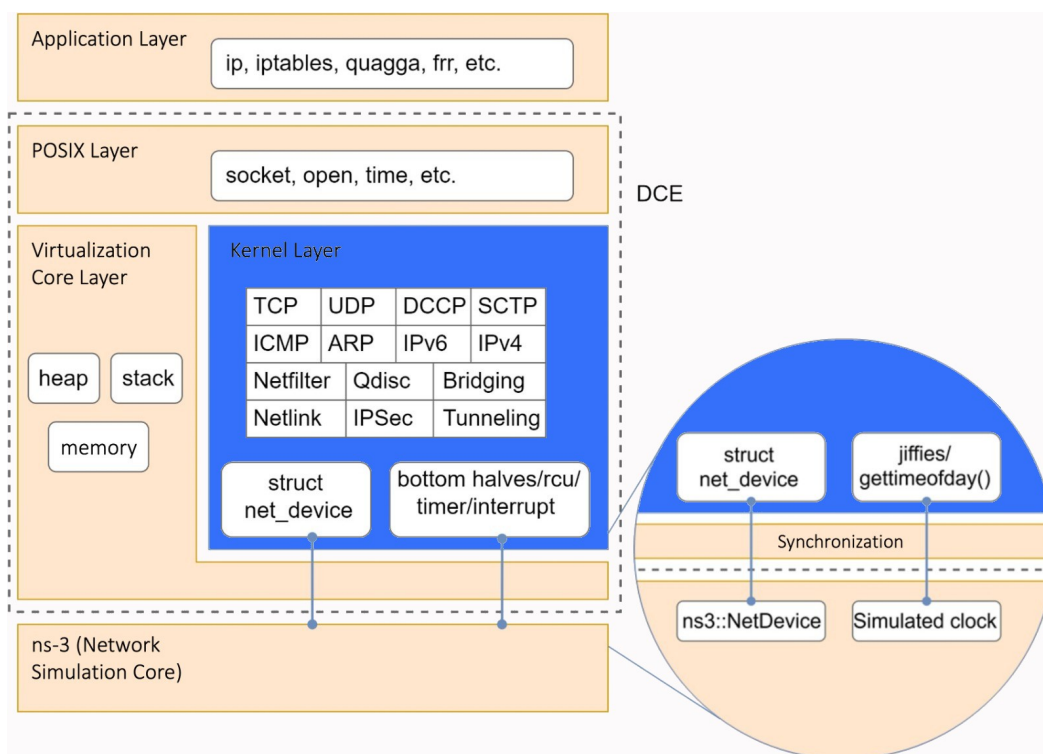


Figure 3. Architecture of DCE. The application layer is where our programs will be executed using DCE to connect to the core of the network simulator (ns-3). [Prepared by the authors on the basis of an image obtained from [25]].

DCE runs each simulated process on the same host process. This model makes it possible to synchronize and schedule each simulated process without having to use inter-process synchronization mechanisms. What’s more, it allows the user to track the behavior of the experiment by different processes without having to use a distributed debugger, which tends to be more complex. The threads in each simulated process are managed by a task handler, implemented in DCE, synchronized with the simulated host and isolated from the other simulated hosts.

Since the loader of the host system aims to ensure that each process does not contain more than one instance of each global variable, DCE provides its own implementation of the loader with a specific loading mechanism to instantiate each global variable, once per simulated instance.

The POSIX implementation in DCE replaces the use of the traditional `glibc` library. Thus, when an application running on top of DCE makes a call to `glibc`, DCE intercepts the call and executes the re-implemented function. Most of these functions are simply a handshake to the corresponding function in the host’s `glibc` library. However, calls that involve system resources must be re-implemented. These include calls involving network resources, the system clock, or memory management. DCE classifies the functions of the library `glibc` using the macros `DCE` or `NATIVE`. The former are functions that are re-implemented by DCE, while the latter are passed to the operating system’s own library.

3.2. Previous Work: Quagga Port

Quagga is a routing software suite, providing implementations of OSPFv2, OSPFv3, RIP v1 and v2, RIPng and BGP-4 for Unix platforms. FRR is a fork of Quagga, which has been embraced by both industry and the community, replacing Quagga as the suite of choice for open source routing projects. FRR incorporates implementations of protocols used in data centers such as OpenFabric [26], and allows the necessary modifications to be incorporated into BGP for routing in large-scale data centers.

Quagga has been ported to DCE in 2008 [27]. The Quagga module in DCE allows using Quagga routing protocols implementation as models in the network simulation. To date, the Quagga DCE project is no longer actively maintained, being its last update in 2012. Despite this, the project is still functional and can be executed with DCE without major problems. Quagga support in DCE is not complete.

To make it easier to use Quagga in simulations, the project provides a `QuaggaHelper` class. This class provides methods that can be used from the simulation scripts to install a protocol on a node and configure it. During the port of FRR, we drew heavily on this class to develop a `FrrHelper` to provide similar facilities. Additionally, the fact that not all features work with the ns-3 stack (Basic Mode) motivated us to focus on the Linux stack (Advanced Mode) for the FRR port.

3.3. DCE Extensions to Support FRR

As previously mentioned, DCE does not support all existing `glibc` functions; therefore, when porting a new application to DCE, it is possible that multiple errors appear due to unrecognized function symbols, since they were not declared in DCE. Therefore, the process of adding support for a new application is very cumbersome, and it is mostly based on trial and error until all needed functions are detected and correctly implemented in DCE.

During the process of porting FRR to DCE, we found several (10) functions that were not declared in the POSIX layer of DCE. For seven of them, it was necessary to implement their functionality inside DCE because they are related to memory allocation, timing, file operations, disk allocation and threading. The other three are functions where their by FRR does not involve system resources, therefore, it is enough to indicate DCE to use the original `glibc` implementation (use the `NATIVE` macro). These extensions can be found in our public repository [22].

In addition to these added functions, we detected two bugs in memory management in functions already implemented under the DCE macro. Two Pull Requests were performed in the ns-3 DCE project due to the correction of these bugs [28,29]. In addition, another Pull Request was made with the necessary functions to execute the code of FRR. At the time of writing this paper, the Pull Requests are pending review.

3.4. FRR Extensions to Run over DCE

In addition to the extensions to DCE mentioned in the previous section, changes were made in FRR in order to run FRR over DCE. These changes were made for practicality reasons, due to the difficulty of adapting DCE to run FRR in its original form. The changes took two forms: changes in the method of compiling and changes in the source code.

In general, in order for an application to run in DCE, it needs to relocate the executable binary into memory. In turn, these executable files need to be built with specific options for the compilation and *linking* stages as explained in the ns-3 DCE manual. For the case of FRR, which is a framework optimized for several different platforms and also for real networking hardware, we need to tune the compiling process.

Compiler optimizations often use function symbols that DCE does not implement. For example, when compiling FRR with the default compiling configuration, the obtained binary uses symbols such as `__strndup` or `pthread_condattr_setclock`. Therefore, we opt to disable some compiler optimization so as to reduce the number of new functions to implement in DCE.

Regarding source code changes to FRR, we perform some minimal modifications to avoid the usage of some unimplemented `glibc`'s function symbols in DCE. The changes are related to the log buffering of FRR, which does not have impact on the functionality of the application. The compiling and source code changes are summarized in a compilation script available in [30].

3.5. Helper Class for Running FRR over DCE

To assist in the creation of simulations using the FRR port, we created a *ns-3-dce-frr* module within the ns-3 DCE project. This is based on the existing *ns-3-dce-quagga* module of the *ns-3-dce-quagga* port.

The *ns-3-dce-frr* module includes, among other things, simulation examples and the `FrrHelper`. The latter contributes to the configuration of the environment required for the deployment of simulations and assists on the instantiation of the selected daemons of FRR where indicated (one or multiple simulated nodes), with zebra being installed implicitly.

Moreover, the `FrrHelper` creates the necessary directories, configuration files and loads the programs to be executed by the nodes. In addition, the `FrrHelper` also includes methods for the configuration of the ported daemons: zebra, BGP and OSPF. Furthermore, a `frr-utils` class has been implemented that provides useful functions for both the `FrrHelper` and the simulations.

3.6. Fat Tree Generator for ns-3 DCE

In order to be able to execute multiple test cases on different configurations of fat-trees, without the need to implement them each time, we develop a fat-tree generator for ns-3 DCE inspired in the Kathará analogous *VFTGen* [18].

This makes it easy to automate, create, and reproduce test cases. To create this generator, the utilities `vftgen-utils` and `vftgen-classes` were implemented, which are responsible for building the topology. That is, according to the indicated parameters, they create the appropriate number of ns-3 nodes, connect them according to the corresponding fat-tree and assign them appropriate IP addresses.

4. Validation and Experimental Results

Our implementation has been evaluated using three different approaches: (i) a simple functional evaluation, (ii) a comparison against the Siybl framework, and (iii) a theoretical analysis.

4.1. Functional Evaluation

Several test cases has been developed along the process of implementing changes and additions to ns-3 DCE to allow the execution of FRR, following an iterative and incremental approach.

The core of FRR architecture is the zebra daemon, which manages IP routing by updating the operating system's kernel routing tables. It also permits to discover interfaces and redistribute routes among the different routing protocols running on the host [31].

Thus, in order to verify the correct functionality of the implementation, it is necessary to run zebra and routing protocols' implementations together; in this case, we focus on BGP. Under normal operation, BGP will learn prefixes and install entries in the kernel routing tables via zebra, allowing the node data plane to forward IP packets. Note that BGP may run without zebra, if we only want to verify the control plane operation (without packet forwarding).

Therefore, for any given scenario, the verification method consists in checking routing table updates. Likewise, connectivity tests can also be performed using, for example, the `ping` command.

For this evaluation, we selected three scenarios and validate the correct execution:

1. Running zebra and BGP in a single node: This scenario allows us to test that FRR with zebra and the chosen routing protocol can be loaded and executed in a node.
2. Running zebra and BGP in a network: We implemented some scenarios based on Kathará project labs [32], such as BGP Simple Peering and BGP Prefix Filtering, which permit us to test BGP update propagation and filtering among peers, and BGP Multi-homed Stub, which is a more complex scenario.
3. Running other routing protocols: Although our focus is on BGP, all the FRR daemons should run on ns-3 DCE. Therefore, we evaluate running OSPF in the same networks

from the previous scenario. It is worth mentioning that no particular change was necessary in the OSPF case, and consequently, we expect that other protocols will also work correctly without the need to make further modifications to the implementation; this is reasonable since, by the architecture of FRR, most of the complexity is contained in the zebra daemon.

After the correct execution of these scenarios, and having verified that the routing tables are correctly updated, we can conclude that zebra and BGP are running correctly in ns-3 DCE, validating our extension.

4.2. Comparison against Sibyl Framework

Given that our focus is on evaluating routing protocols over Clos networks, we decided to validate our implementation running several experiments over the same network topologies used by the Sibyl framework [9]. Using these same scenarios in ns-3 DCE gave us the opportunity to compare the results, since Sibyl framework defines various fat-tree based scenarios.

4.2.1. Experimental Setup

The experiments consist on recording the number of Protocol Data Units (PDUs) exchanged among nodes until convergence of the routing protocol. In our case of study (the BGP protocol), this is equivalent to the number of BGP UPDATE messages. It worth noting that each experiment consider the propagation of a single network prefix for each leaf node in the topology, without losing generality in the results.

When a simulation starts and the BGP protocol begins to execute, the nodes start exchanging UPDATE messages, until convergence is reached and the UPDATE messages cease to be exchanged. Convergence occurs when all topology information has been distributed (i.e., multi-path connectivity has been reached for every network prefix). Any change in the topology, or in the routing table of a node, generates a new exchange of UPDATE messages until convergence is reached again.

A scenario comprises a certain fat-tree topology (determined by the parameters k_{leaf} , k_{top} and redundancy) and five different situations (test cases) that are described below:

- **Bootstrap:** The objective is to study the standard behavior of the protocol in the topology when it is started, without any failure.
- **Node Failure:** This test case is used both to verify that BGP converges after a switch failure, and also to count the number of PDUs that the protocol exchanged for that purpose. The fault can be introduced in any type of switch in the topology, that is, Leaf, Spine or Tof. It is done by shutting down the BGP daemon on the given switch.
- **Node Recovery:** In this test case, the objective is to count the number of PDUs exchanged by the switches, after one of them fails and is replaced by a new one. Like the previous case, this case can be run on a Leaf, Spine, or Tof. We implement this case by raising the topology without running BGP on the node in question, we wait for it to converge and then we start BGP. This is equivalent to crashing the node and then starting it again.
- **Link Failure:** This case also has two goals. On the one hand, to verify BGP convergence after a link failure, and on the other hand, to count the number of PDUs for this purpose. The test can be run for both the Leaf–Spine link case and the Spine–Tof link case, simply by pulling down a given interface.
- **Link Recovery:** This case counts the number of PDUs after a failed link is replaced. That is, the simulation is started and the protocol is expected to converge. Link failure is then caused and the protocol is again expected to converge. Finally, the link is recovered and the new convergence is expected. The number of PDUs that are taken into account are those exchanged in this last phase.

Each scenario is named using the following criterion: $x_y_z_case-level$, where:

- x is the k_leaf parameter.
- y is the k_top parameter.
- z is the $redundancy$ parameter.
- $case$ represents the test case, which can be `link-failure`, `link-recovery`, `node-failure`, or `node-recovery`.
- $level$ depends on the case:
 - If the case is `link-failure` or `link-recovery`, $level$ can be `leaf-spine` or `spine-tof`, referencing the level where link failure or recovery occurred.
 - If the case is `node-failure` or `node-recovery`, $level$ can be `leaf`, `spine` or `tof`, referencing the level where the failure or recovery of the node occurred.

The different scenarios were configured using the same values for k_leaf and k_top parameters so as to have homogeneous switches at the different levels of the fat-trees. On the other hand, for a given value of k_leaf and k_top , we vary the $redundancy$ parameter, always considering that it divides the k_top value. Moreover, during the different executions of the test cases, we vary the level where the failure is produced so as to cover all the possibilities.

4.2.2. Execution Environment

All the experiments presented were executed on a server machine running Ubuntu 16.04 with 30 CPUs AMD Opteron 63xx class and with 244 GB of RAM memory.

We configure the simulation duration so as to allow convergence while minimizing it. For this, we studied several simulations to find the best values for each scenario. The final configurations for the simulation duration (in simulated time) for each scenario are:

- Bootstrap: 10 s.
- Node-failure and link-failure: 20 s. This time allows for the bootstrap to finish and converge, produce the failure in the node or link and then wait again for convergence.
- Node-recovery and link-recovery: 30 s. In this case, after the failure and the convergence, the node or link is recovered and we have to wait again for convergence.

The simulations are configured to generate traffic capture files (`.pcap`) for every interface of each node simulated. These files are then processed so as to count the number of BGP's UPDATE messages exchanged.

4.2.3. Results

The results of all the experiments executions are shown in Appendix A. As can be seen from Table A1, the number of PDUs (BGP UPDATES) obtained with our simulations in ns-3 DCE exactly match the number obtained with the Sibyl emulations for most of the scenarios. This exact match between the results in ns-3 DCE and the emulation platform strongly validates the accuracy of our simulation platform. In particular, this shows that with the proposed platform, we can execute the exact same BGP algorithm that runs in the Sibyl emulation approach.

Regarding the scenarios where there are differences, we should note that there are some Sibyl scenarios that present more than one result. This is due to the fact that the emulations are not deterministic, and depend, for example, on the host machine resource usage. For the cases where we have more than one result from Sibyl, the results obtained in our simulations are between these values or very close to them. These differences also demonstrate one of the main advantages of the simulation against the emulation given that in the simulation the results are deterministic and reproducible. The exact same result can be obtained independently of the underlying hardware or software where it is run.

In Table 1, we select some specific results. In particular, we show two cases with a significant difference between our experiments and the Sibyl framework results. If we consider the couple of scenarios painted in blue in Table 1, the result in scenario `10_10_1_node-failure-spine` for Sibyl is roughly half the result obtained in ns-3 DCE;

we argue that this is an outlier in Sibyl, as we will further show in Section 4.3.1. Regarding 12_12_1_node-failure-leaf scenario, note that the result for Sibyl is smaller, but in the same order than the one of ns-3 DCE; here, we argue that the vector-distance nature of BGP and its well known characteristic of “path hunting” is responsible for this difference, as we further explain in Section 4.3.2.

Table 1. Representative results for comparison against Sibyl.

Scenario	Number of nodes	PDUs ns-3	PDUs ns-3	PDUs Sibyl
		w/o zebra	w zebra	
2_2_1_node-failure-leaf	20	60	60	60
2_2_2_node-failure-leaf	10	28	28	28
4_4_1_node-failure-leaf	80	504	504	504
4_4_2_node-failure-leaf	40	248	248	248
4_4_4_node-failure-leaf	20	120	120	120
6_6_1_link-failure-leaf-spine	180	996	996	996
6_6_1_link-recovery-spine-tof	180	1796	1796	1657, 1796
10_10_1_node-failure-spine	500	37,480	37,480	18,390
12_12_1_node-failure-leaf	720	17,244	-	13,800

Note that we can also compare the results of the simulations with and without the zebra daemon running. As explained in Section 5, we executed the experiments disabling zebra in order to reduce the resource consumption of the simulation. In most of the scenarios considered, this change does not affect the number of BGP updates exchanged in the experiment. Nevertheless, in some cases there exists a small difference (actually, in 8 out of 250 experiments, 3.2%). Most of the misalignment in the results are experienced in node-failure-leaf scenarios, due to the very reason we mentioned above: the path-vector nature of BGP may cause extra UPDATE messages to be exchanged, as explained in Section 4.3.2.

In the following section, we also compare the obtained results against a theoretical model of the behavior of BGP.

4.3. Theoretical Analysis of BGP Behavior over Selected Scenarios

In this section, we will analyze the BGP behavior over the scenarios presented before, taking advantage of the regularity of the fat-trees multi-plane with $R = 1$, which can be described by a single parameter k [33]. In effect, in a fat-tree topology of k PoDs, there are k switches (each with k ports) in each PoD, arranged in two levels (Leafs and Spines) of $k/2$ switches each. Each Leaf is connected to the $k/2$ Spines and vice versa. There are $(k/2)^2$ Core switches, each of which connects to k PoDs.

The aim of this theoretical analysis is to find an expression in function of k that describes the number of packets exchanged in two scenarios: the fail of a leaf, and the fail of a spine. These scenarios were intentionally selected after the differences in results shown in Table 1.

Remember that the experiments propagate one prefix per leaf, i.e., in a fat-tree of k PoDs there are a total of $k^2/2$ prefixes, or equivalently, leaf nodes.

4.3.1. Case Spine Node Failure

To analyze the behavior of BGP when a spine node fails, we divided the problem into three sub-problems: (1) the PoD of the failure, (2) the PoD with no failures and (3) the spine-core links. Since the goal is to find an expression that models the total number of packets exchanged after the failure, dividing the problem into sub-problems is equivalent to dividing the expression into sums.

1. First note that each leaf needs connectivity information for $k^2/2$ prefixes; while $k^2/2 - 1$ prefixes are “foreign”, the remaining one is directly connected. When the fail occurs, in the PoD of the fail there are $k/2$ leaf nodes aware of the failure. The BGP process of each leaf node will recalculate the routes and will notice that for every known prefix, one possible next-hop is missing. Consequently, it will send, for each known prefix, a BGP update with the next-hop attribute updated. Thus, we have $k/2$ leafs sending $k^2/2 - 1$ packets (the total number of prefixes in the fabric which have lost a next-hop) through their $k/2 - 1$ links. Consequently, the total amount of BGP packets in the PoD of the failure equals $k/2 \times (k/2 - 1) \times (k^2/2 - 1)$.
2. The rest of the PoDs learn about the failure through the spine connected to the same plane as the faulty spine, through the corresponding core switch. This spine sends a BGP withdraw containing all the prefixes no longer reachable through the corresponding core switch (all the prefixes inside the PoD with the failure) to all its neighbors ($k/2$ leaves in this PoD). After that, each leaf recalculate its routes and notice that each prefix received in the withdraw are no longer reachable through one of its next hops. Consequently, it will send, for each of these $k/2$ prefixes, a BGP update to all its neighbors ($k/2$ spines). Consequently, the total amount of BGP packets in each PoD without a failure equals $(k/2) + (k/2) \times (k/2) \times (k/2)$.
3. The faulty spine was connected to $k/2$ core switches. Because the topology considered is multi-plane with $R = 1$, these core nodes have exactly one link with each PoD. After the failure, each core connected with the faulty spine have no longer reachability to the prefixes of the corresponding PoD, and it must send a BGP withdraw for the prefixes of such PoD to all its neighbors ($k - 1$ spines). After that, when a spine connected to these cores receives the withdraw from all of them, it will notice that it no longer has reachability to the prefixes of the given PoD, and it will send the correspondent withdraw upstream to the cores; therefore, a total of two BGP packets traverse every core–spine link. In effect, we have $k/2$ cores, which send and receive one BGP withdraw through all their “live” interfaces ($k - 1$). Consequently, the total amount of BGP packets in the core–spine links equals $2 \times (k - 1) \times k/2 = (k - 1) \times k$ packets.

Put together, and multiplying the expression for the PoDs without a failure times the amount of such PoDs ($k - 1$), we arrive at a total BGP packets of $k/2 \times (k/2 - 1) \times (k^2/2 - 1) + (k - 1) \times ((k/2) + (k/2) \times (k/2) \times (k/2)) + (k - 1) \times k$. Simplifying, we obtain the polynomial

$$\frac{k^4}{4} - \frac{3k^3}{8} + \frac{5k^2}{4} - k$$

Figure 4 compares packet growth as a function of the number of nodes for the results of ns-3, Sibyl and the polynomial expression. Notice that the results of ns-3 fits the polynomial exactly. On the other hand, Sibyl results shows a deviation from the polynomial, and there are far less results for this particular case. In fact, the Sibyl results for $k = 20$ double those expected following the theoretical expression. If we look closely at the presented analysis (step 1), the Sibyl packet count for this case barely exceeds the number of packets needed to update the routes within the PoD of the failure. Therefore, our assumption that the difference for the scenario 10_10_1_node-failure-spine shown in Table 1 is an outlier is confirmed.

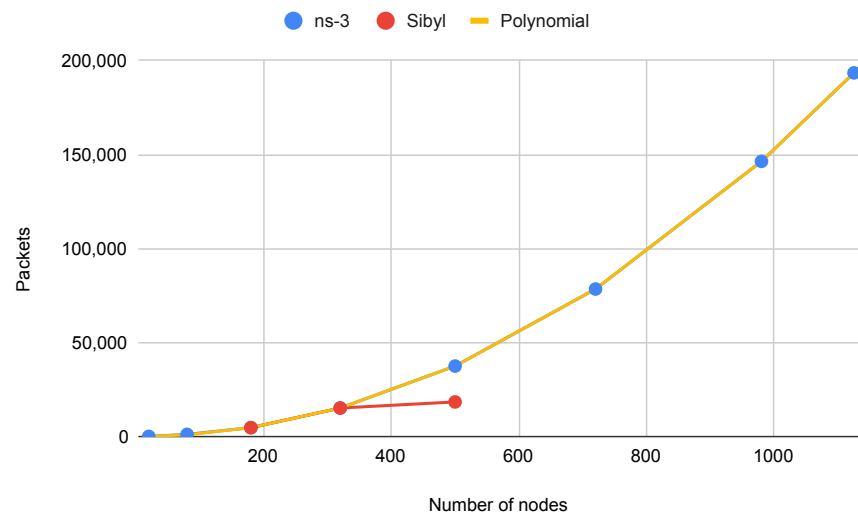


Figure 4. Evolution of the results of ns-3 and Sibyl for the Spine failure scenario, in comparison with the evolution of the polynomial $\frac{k^4}{4} - \frac{3k^3}{8} + \frac{5k^2}{4} - k$.

4.3.2. Case Leaf Node Failure

To analyze this case, let us consider what happens at the routing level when a leaf fails; this is analogous to a prefix that is no longer reachable, and $k/2$ links down in the PoD of the failure. When a leaf is no longer reachable, its neighbors ($k/2$ spines of the given PoD) will notify the fact with a BGP withdrawn that will spread throughout the fabric. In terms of packet count, this implies that each node in the fabric will send a BGP withdrawn out all of its interfaces. Similarly, two packets must be observed on each fabric link.

The total number of links of a fat-tree multi-plane with $R = 1$ are $k^3/2$ (before the leaf failure). As mentioned, $k/2$ links go down after the failure, and consequently, if each link carries two packets, the total amount of BGP packets for a leaf failure is $2 \times (k^3/2 - k/2)$ or equivalently

$$k^3 - k \tag{1}$$

Note that this is a lower bound due to the following. When a Leaf node receives from a Spine the aforementioned withdrawal, its routing table still holds the reachability information for the given prefix using the rest of the spines, and therefore the leaf “thinks” it can still reach the prefix. This race condition can cause the leafs to send an BGP Update announcing the (now inexistent) routes to its corresponding spines in the PoD. Every announced route contains in its AS-PATH the ASN of the spine that receives it, and therefore is discarded by it. Thanks to the specific numbering of ASNs in the fat-tree, the inexistent route is no longer propagated, and the “path hunting” is stopped early. A simple way to find an expression that models that behavior is by adding one more packet for every spine–leaf link to the above expression (1). To determine the number of links, first we count the number of links inside a normal PoD, i.e., $k/2 \times k/2$ and multiply this by the amount of normal PoDs ($k - 1$). Then, the number of links in the PoD of the failure is $k/2 \times (k/2 - 1)$. Consequently, the total amount of BGP packets that model this behavior is

$$\frac{5k^3}{4} - \frac{3k}{2} \tag{2}$$

Figure 5 compares packet growth as a function of k for the results of ns-3, Sibyl and the polynomials expressions for the Leaf Node failure scenario. Note that while Sibyl results follow polynomial 1, ns-3 DCE results follow both 1 and 2 alternatively. Regardless, the results are correct since both behavior may occur due to the nature of BGP and timing of control plane packets.

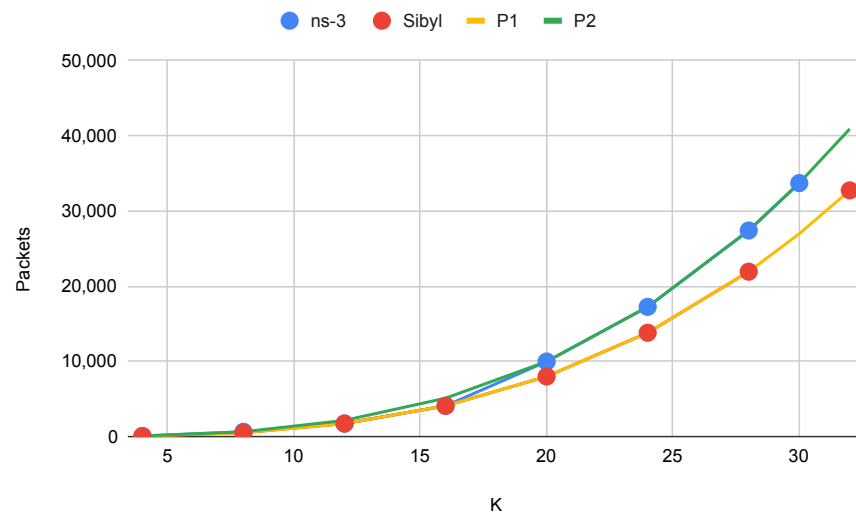


Figure 5. Evolution of the results of ns-3 and Sibyl for the Leaf failure scenario, in comparison with the evolution of the polynomials $P1 = k^3 - k$ and $P2 = \frac{5k^3}{4} - \frac{3k}{2}$.

5. Performance Analysis

Given the possible massive size of a network (in number of nodes and links) in MSDCs, it is important to study how the simulator performs and scales with different network's sizes.

Specifically, we decided to evaluate the scalability in two aspects, memory usage and simulation runtime for different network sizes.

Even more, while developing and testing our implementation, we notice that the performance regarding memory usage and simulation time was a limiting factor to obtain results with big networks. Therefore, we decided to incorporate two features to improve the performance: disable the zebra daemon and disable IPv6 support.

Regarding the first one, our implementation allows us to run a simulation without running the zebra daemon. In fact, a routing protocol in FRR does not need zebra to execute, therefore it can be tested correctly without using zebra. On the other hand, when disabling zebra, we lost the data plane functionality, and the kernel routing tables are not updated. This means that the nodes will exchange routing information following the routing protocol algorithm, but the routes will not be saved. As a consequence, there will not be connectivity between nodes.

Nevertheless, as our objective is to study the behavior of the control plane only, in particular the convergence of BGP under different situations, the previous drawbacks do not affect our results.

So as to be able to run a routing protocol (and BGP in particular) without zebra, some modifications and extensions are needed. For example, it is not possible to discover neighbor interfaces. For the case of BGP, this implies that in the configuration it is not possible to refer to the *BPG peers* in a generic form using the interface. Instead, we must use the IP address. In our code, we provide all the configurations and modifications of the `FrrHelper` to run simulations with the zebra daemon disabled.

Regarding IPv6, we notice a high load of IPv6 control packets such as *ICMPv6 Router Solicitation* and *ICMPv6 Router Advertisement* in our simulations. This was due to the use of the Linux kernel in DCE, which includes IPv6 in all interfaces by default. Therefore, our solution allows us to easily disable the use of IPv6 in all interfaces by executing a single command.

Additionally, in the results we also show the execution time of each scenario. It can be seen that the experiments without zebra can run faster, taking approximately half of the time to execute in comparison with the experiments with zebra. Even more, when

running without zebra, the simulations use much less RAM memory, which allows us to experiment with bigger scenarios. Without zebra, we were able to run a scenario with 1125 nodes (15_15_1_*) while when using zebra, the biggest scenario was one with 320 nodes (8_8_1_*).

In summary, if the objective of a given use case is only to test the control plane of a routing algorithm, it is recommended to disable the zebra daemon, as it has shown to improve the simulations performance in execution time and consumed memory.

In order to illustrate the performance improvement that the above changes produced, the example 6_6_6_link-failure-spine-tof was run. In it, the changes will be applied independently, and then all together. Table 2 illustrates the results of execution time, memory used and improvement percentage for the introduced change.

Table 2. Results of execute the scenario 6_6_6_link-failure-spine-tof after applying performance improvements.

Proposed feature	Execution time	Memory consumption	Execution time improvement
None	2:50.140	705 MB	-
Disable IPv6	1:44.284	705 MB	+38.71%
Reduce simu time	2:18.452	705 MB	+18.62%
Without zebra	1:29.786	403 MB	+47.23%
All together	1:18.833	403 MB	+53.67%

This improved version allows us to scale the experiments and perform the same scenarios performed with Sibyl in the work of reference [9]. Despite both experimentation environments managing the internal time differently, we can still compare the execution times and the resource consumption of each environment. These comparisons, for scenarios implementing the densest fat-tree topologies, are shown in Table 3. The resources available for the ns-3 environment are the presented in Section 4.2.2, i.e., 30 CPUs and 244 GB of RAM. On the other hand, the scenarios 2_2_1 to 8_8_1 from Sibyl were executed on a cluster of 22 VMs, each with 2-core vCPUs and 8 GB of vRAM, while the scenarios 10_10_1 to 16_16_1 were executed on a cluster composed of 160 VMs, each with 4-core vCPUs and 8 GB vRAM.

It is worth noting that our environment use a single CPU, this is due to the nature of the ns-3 simulator with DCE. Additionally, in addition to the information provided in the table, we observed a RAM saturation in the largest scenario performed. This can explain the gap between the execution time presented in the last of the simulated scenarios in ns-3.

Table 3. Comparison of execution times between the environment in ns-3 and the sibyl framework for the failure of a leaf node in fat-trees multi-plane.

Scenario	Number of nodes	Execution time in ns-3	Execution time in Sibyl
2_2_1_node-failure-leaf	20	0:59	2:2
4_4_1_node-failure-leaf	80	4:12	3:4
6_6_1_node-failure-leaf	180	10:41	6:25
8_8_1_node-failure-leaf	320	21:16	20:49
10_10_1_node-failure-leaf	500	36:31	14:18
12_12_1_node-failure-leaf	720	1:22:40	37:3
14_14_1_node-failure-leaf	980	2:41:23	1:5:31
15_15_1_node-failure-leaf	1125	7:21:43	-
16_16_1_node-failure-leaf	1280	-	1:44:58

6. Discussion and Conclusions

As a general result, we can conclude that the FRR port to ns-3 DCE is functionally correct and promisingly scalable. In fact, running control plane-only experiments (i.e., with-

out the zebra daemon) on a single server, we were able to achieve competitive results as Sibyl running on a cluster of computing nodes.

In terms of execution times, the results were very satisfactory. ns-3 DCE shows a performance almost close to the Sibyl-emulated environment for scenarios up to 320 nodes, which were run on infrastructures with similar resources. On the other hand, for scenarios with more than 320 nodes, ns-3 DCE shows an average time 2.4 times higher than Sibyl, but with a cumulative vRAM ratio of 5.2 times lower.

The port validation included a theoretical analysis of the behavior of BGP for the datacenter on multi-plane fat-tree topologies. This analysis exposes how to find a formal expression that describes the growth of control packets injected into the network after a failure scenario. Although this analysis validated the results obtained, it is worth noting that the implementations of the routing protocols may be subject to race conditions or limited by the available resources that slightly vary the behavior based on optimizations of the implementations.

In this paper, we focused on BGP in the datacenter, and briefly commented about the execution of other routing daemons of the FRR suite. In this regard, another straightforward line of future work is to undertake a thoroughly testing of other routing daemons, in principle in the MSDC scope. To this end, Openfabric (IS-IS with flooding reduction) is already implemented and ready to run.

Overall, to the best of our knowledge, in this work we provide a functionally correct and scalable FRR port to ns-3 DCE, ready to use by researchers and practitioners alike. For the time being, we only focused on the control plane of Fat-Tree network routing protocols, reaching competitive results with less resource consumption. A foreseeable line of research shall include the forwarding plane, enabling research on traffic behavior in MSDC and/or other topologies.

Author Contributions: Conceptualization, L.A., E.G., M.R., S.A. and F.V.; methodology, L.A., E.G., M.R.; software, S.A. and F.V.; validation, S.A., F.V. and L.A.; formal analysis, L.A., E.G. and M.R.; investigation, L.A., E.G. and M.R.; writing—original draft preparation, L.A., E.G., M.R., S.A. and F.V.; writing—review and editing, L.A., E.G. and M.R.; visualization, L.A., S.A. and F.V.; supervision, E.G. and M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the Uruguayan National Research and Innovation Agency (ANII) under Grant No. POS_NAC_M_2020_1_163847.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: We would like to thank our colleagues from the Computer Networks research group at Università Roma Tre, Italy, for the facilitation of execution times reported by the Sibyl framework in their environment.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Table A1. Comparison of execution time and number of PDUs exchanged between the proposed ns-3 simulation platform and Sibyl for all the scenarios.

Scenario	Number of nodes	Execution time w/o zebra	PDUs ns-3 w/o zebra	Execution time w zebra	PDUs ns-3 w zebra	PDUs Sibyl
2_2_1_link-failure-leaf-spine	20	1:07.19	44	1:18.73	44	-
2_2_1_link-failure-spine-tof	20	0:53.28	45	1:30.77	45	-
2_2_1_link-recovery-leaf-spine	20	0:54.39	69	1:20.07	69	-
2_2_1_link-recovery-spine-tof	20	1:04.59	72	1:40.31	72	-
2_2_1_node-failure-leaf	20	0:59.59	60	1:24.61	60	60
2_2_1_node-failure-spine	20	1:01.74	56	1:40.42	56	-
2_2_1_node-failure-tof	20	1:02.40	72	1:21.33	72	-
2_2_1_node-recovery-leaf	20	0:52.62	96	1:03.42	96	-
2_2_1_node-recovery-spine	20	0:58.33	160	1:50.73	160	-
2_2_1_node-recovery-tof	20	0:55.88	168	1:19.57	168	-
2_2_2_link-failure-leaf-spine	10	0:37.19	16	0:46.51	16	-
2_2_2_link-failure-spine-tof	10	0:30.20	12	0:52.36	12	-
2_2_2_link-recovery-leaf-spine	10	0:33.69	29	0:32.79	29	-
2_2_2_link-recovery-spine-tof	10	0:32.03	26	0:32.48	24	-
2_2_2_node-failure-leaf	10	0:34.74	28	0:50.99	28	28
2_2_2_node-failure-spine	10	0:32.22	18	0:36.68	18	-
2_2_2_node-failure-tof	10	0:34.83	24	0:50.47	24	-
2_2_2_node-recovery-leaf	10	0:32.05	48	0:48.94	48	-
2_2_2_node-recovery-spine	10	0:34.10	68	0:47.40	68	-
2_2_2_node-recovery-tof	10	0:35.70	72	0:32.06	72	-
4_4_1_link-failure-leaf-spine	80	4:15.53	312	5:23.27	312	-
4_4_1_link-failure-spine-tof	80	4:15.09	427	5:10.87	427	-
4_4_1_link-recovery-leaf-spine	80	4:37.03	409	5:49.97	409	-
4_4_1_link-recovery-spine-tof	80	4:23.28	542	5:33.42	542	-
4_4_1_node-failure-leaf	80	4:12.73	504	5:17.12	504	504
4_4_1_node-failure-spine	80	4:28.11	1128	5:57.28	904	-
4_4_1_node-failure-tof	80	4:14.70	1568	5:39.91	1568	-
4_4_1_node-recovery-leaf	80	4:31.55	768	6:09.64	768	-
4_4_1_node-recovery-spine	80	4:13.09	1808	6:00.58	1808	-
4_4_1_node-recovery-tof	80	4:16.72	2320	6:11.65	2320	-

Table A1. Cont.

Scenario	Number of nodes	Execution time w/o zebra	PDU ns-3 w/o zebra	Execution time w zebra	PDU ns-3 w zebra	PDU Sibyl
4_4_2_link-failure-leaf-spine	40	2:11.22	96	3:03.51	96	-
4_4_2_link-failure-spine-tof	40	2:07.43	112	3:15.35	112	-
4_4_2_link-recovery-leaf-spine	40	2:20.26	145	3:29.08	145	-
4_4_2_link-recovery-spine-tof	40	2:07.44	162	3:10.86	162	-
4_4_2_node-failure-leaf	40	2:10.24	248	3:08.23	248	248
4_4_2_node-failure-spine	40	2:13.89	292	3:00.28	292	-
4_4_2_node-failure-tof	40	2:06.92	672	2:59.16	672	-
4_4_2_node-recovery-leaf	40	2:09.55	384	3:11.17	384	-
4_4_2_node-recovery-spine	40	2:11.83	640	3:11.03	640	-
4_4_2_node-recovery-tof	40	2:00.63	1040	3:22.35	1040	-
4_4_4_link-failure-leaf-spine	20	1:05.64	72	1:16.44	72	-
4_4_4_link-failure-spine-tof	20	1:03.96	56	1:31.83	56	-
4_4_4_link-recovery-leaf-spine	20	1:12.47	97	1:19.03	97	-
4_4_4_link-recovery-spine-tof	20	1:10.85	78	1:32.06	78	-
4_4_4_node-failure-leaf	20	0:58.07	120	1:44.26	120	120
4_4_4_node-failure-spine	20	1:08.24	196	0:59.93	196	-
4_4_4_node-failure-tof	20	1:07.31	224	1:30.04	224	-
4_4_4_node-recovery-leaf	20	1:03.67	192	1:15.28	192	-
4_4_4_node-recovery-spine	20	1:09.48	384	1:49.51	384	-
4_4_4_node-recovery-tof	20	1:06.60	400	0:59.46	400	-
6_6_1_link-failure-leaf-spine	180	10:42.61	996	14:40.34	996	996
6_6_1_link-failure-spine-tof	180	10:27.84	1529	14:17.19	1529	1529, 1577
6_6_1_link-recovery-leaf-spine	180	11:11.31	1213	15:53.67	1213	1069, 1238
6_6_1_link-recovery-spine-tof	180	10:55.97	1796	16:14.04	1796	1657, 1796
6_6_1_node-failure-leaf	180	10:41.58	1716	13:52.94	1716	1716, 1734
6_6_1_node-failure-spine	180	10:12.82	4704	15:13.56	4704	4704, 4726
6_6_1_node-failure-tof	180	10:19.49	8712	14:31.77	8712	8712, 8808
6_6_1_node-recovery-leaf	180	10:38.73	2592	14:44.55	2592	2592, 3444
6_6_1_node-recovery-spine	180	10:23.92	7872	15:32.47	7872	8686, 9362
6_6_1_node-recovery-tof	180	9:56.09	11256	14:51.07	11256	11190, 11316

Table A1. Cont.

Scenario	Number of nodes	Execution time w/o zebra	PDU ns-3 w/o zebra	Execution time w zebra	PDU ns-3 w zebra	PDU Sibyl
6_6_2_link-failure-leaf-spine	90	5:08.22	288	7:25.55	288	288, 358
6_6_2_link-failure-spine-tof	90	5:03.01	396	7:19.56	396	396, 456
6_6_2_link-recovery-leaf-spine	90	5:26.72	397	7:34.14	397	325, 397
6_6_2_link-recovery-spine-tof	90	5:26.11	500	7:43.13	500	439, 506
6_6_2_node-failure-leaf	90	5:16.17	852	7:12.16	852	852, 858
6_6_2_node-failure-spine	90	5:09.56	1446	8:01.95	1446	1446
6_6_2_node-failure-tof	90	5:05.01	3960	7:21.65	3960	3960
6_6_2_node-recovery-leaf	90	4:49.43	1296	7:27.46	1296	1296, 1926
6_6_2_node-recovery-spine	90	5:09.29	2580	7:21.40	2580	2916, 3247
6_6_2_node-recovery-tof	90	5:13.78	5208	7:29.28	5208	5208, 5310
6_6_3_link-failure-leaf-spine	60	3:13.01	228	5:07.65	228	228
6_6_3_link-failure-spine-tof	60	3:32.99	264	4:34.69	264	264
6_6_3_link-recovery-leaf-spine	60	3:31.02	301	5:07.55	301	253, 301
6_6_3_link-recovery-spine-tof	60	3:46.15	332	5:25.22	332	295, 338
6_6_3_node-failure-leaf	60	3:28.60	564	5:30.40	564	562, 564
6_6_3_node-failure-spine	60	3:27.40	1086	4:29.95	1086	1086
6_6_3_node-failure-tof	60	3:32.06	2376	4:37.03	2376	2376, 2942
6_6_3_node-recovery-leaf	60	3:26.27	864	4:54.17	864	864, 1002
6_6_3_node-recovery-spine	60	3:24.86	1860	5:41.32	1860	1836, 2413
6_6_3_node-recovery-tof	60	3:25.22	3192	5:08.39	3192	3354, 3450
6_6_6_link-failure-leaf-spine	30	1:46.37	168	2:14.63	168	168, 541
6_6_6_link-failure-spine-tof	30	1:44.24	132	2:37.48	132	132
6_6_6_link-recovery-leaf-spine	30	1:48.37	205	2:29.47	205	181, 205
6_6_6_link-recovery-spine-tof	30	1:47.56	164	2:32.80	164	151, 170
6_6_6_node-failure-leaf	30	1:43.67	276	2:26.95	276	276
6_6_6_node-failure-spine	30	1:47.86	726	2:39.75	726	726
6_6_6_node-failure-tof	30	1:44.04	792	2:29.69	792	792
6_6_6_node-recovery-leaf	30	1:42.01	432	3:08.22	432	432
6_6_6_node-recovery-spine	30	1:46.04	1140	2:28.39	1140	1104, 1182
6_6_6_node-recovery-tof	30	1:45.79	1176	2:28.30	1176	1176

Table A1. Cont.

Scenario	Number of nodes	Execution time w/o zebra	PDU ns-3 w/o zebra	Execution time w zebra	PDU ns-3 w zebra	PDU Sibyl
8_8_1_link-failure-leaf-spine	320	21:56.54	2288	29:13.19	2288	2288
8_8_1_link-failure-spine-tof	320	20:06.32	3735	29:26.26	3735	3743
8_8_1_link-recovery-leaf-spine	320	22:56.82	2673	35:14.27	2673	2417
8_8_1_link-recovery-spine-tof	320	22:28.77	4218	34:43.00	4218	4218
8_8_1_node-failure-leaf	320	21:16.01	4080	30:44.58	4080	4076, 4080, 4088
8_8_1_node-failure-spine	320	21:12.30	15152	30:26.00	15152	15152, 15197
8_8_1_node-failure-tof	320	20:56.05	28800	28:37.21	28800	28958, 29058
8_8_1_node-recovery-leaf	320	21:26.21	6144	33:47.08	6144	8172, 10228
8_8_1_node-recovery-spine	320	21:09.10	22816	31:55.63	22816	22848
8_8_1_node-recovery-tof	320	21:26.99	34848	30:46.45	34848	34622, 35076
8_8_2_link-failure-leaf-spine	160	9:51.76	640	15:02.54	640	640
8_8_2_link-failure-spine-tof	160	9:52.78	960	14:14.68	960	960
8_8_2_link-recovery-leaf-spine	160	10:59.49	833	16:07.96	833	705, 855
8_8_2_link-recovery-spine-tof	160	10:33.65	1146	14:20.73	1146	1033, 1154
8_8_2_node-failure-leaf	160	10:28.93	2032	14:30.95	2032	2028, 2032
8_8_2_node-failure-spine	160	9:56.79	4488	14:38.23	4488	4488, 4510
8_8_2_node-failure-tof	160	10:18.38	13440	14:32.53	13440	13440, 13680
8_8_2_node-recovery-leaf	160	10:15.41	3072	15:36.29	3072	4584
8_8_2_node-recovery-spine	160	10:05.70	7136	14:24.24	7136	7104, 10439
8_8_2_node-recovery-tof	160	9:38.98	16416	13:43.01	16416	16599, 16832
8_8_4_link-failure-leaf-spine	80	4:53.83	416	7:01.98	416	416, 447
8_8_4_link-failure-spine-tof	80	4:51.56	480	7:02.12	480	480
8_8_4_link-recovery-leaf-spine	80	5:18.90	513	7:16.01	513	449, 513
8_8_4_link-recovery-spine-tof	80	5:14.52	570	7:09.06	570	521, 578
8_8_4_node-failure-leaf	80	5:06.75	1008	6:52.72	1008	1008, 1024
8_8_4_node-failure-spine	80	5:02.55	2696	6:27.52	2696	2696
8_8_4_node-failure-tof	80	4:59.97	5760	6:39.84	5760	5760
8_8_4_node-recovery-leaf	80	4:57.09	1536	6:42.37	1536	1784, 2278
8_8_4_node-recovery-spine	80	4:52.50	4064	7:18.56	4064	4400, 4802
8_8_4_node-recovery-tof	80	5:04.17	7200	6:54.38	7200	7200

Table A1. Cont.

Scenario	Number of nodes	Execution time w/o zebra	PDU _s ns-3 w/o zebra	Execution time w zebra	PDU _s ns-3 w zebra	PDU _s Sibyl
8_8_8_link-failure-leaf-spine	40	2:31.42	304	3:19.37	304	304
8_8_8_link-failure-spine-tof	40	2:02.31	240	3:03.47	240	240
8_8_8_link-recovery-leaf-spine	40	2:04.05	353	3:59.65	353	321, 353
8_8_8_link-recovery-spine-tof	40	1:54.87	282	3:36.25	282	265, 290
8_8_8_node-failure-leaf	40	2:05.69	496	3:45.87	496	496
8_8_8_node-failure-spine	40	1:56.52	1800	3:30.05	1800	1800
8_8_8_node-failure-tof	40	2:16.27	1920	3:16.83	1920	1920
8_8_8_node-recovery-leaf	40	2:07.44	768	3:48.72	768	768, 888
8_8_8_node-recovery-spine	40	2:21.66	2528	3:23.44	2528	2613, 2856
8_8_8_node-recovery-tof	40	2:09.00	2592	3:43.65	2592	2592, 2960
10_10_1_link-failure-leaf-spine	500	43:21.57	4380	53:33.93	4380	-
10_10_1_link-failure-spine-tof	500	35:48.74	7429	52:05.10	7429	-
10_10_1_link-recovery-leaf-spine	500	40:25.85	4981	1:03:15	4981	-
10_10_1_link-recovery-spine-tof	500	38:10.10	8192	59:07.93	8192	-
10_10_1_node-failure-leaf	500	36:31.21	7980	54:41.84	7980	7980
10_10_1_node-failure-spine	500	37:16.85	37480	53:23.76	37480	18390
10_10_1_node-failure-tof	500	35:52.55	72200	55:32.04	72200	36072
10_10_1_node-recovery-leaf	500	40:15.09	12000	55:27.11	12000	-
10_10_1_node-recovery-spine	500	36:11.80	52640	58:46.43	52640	-
10_10_1_node-recovery-tof	500	37:35.48	84040	54:39.94	84040	-
10_10_2_link-failure-leaf-spine	250	16:59.57	1200	23:04.10	1200	1200
10_10_2_link-failure-spine-tof	250	17:04.08	1900	26:00.38	1900	1900
10_10_2_link-recovery-leaf-spine	250	18:30.72	1501	26:21.99	1501	1301, 1501
10_10_2_link-recovery-spine-tof	250	18:25.44	2192	27:09.12	2192	2011, 2202
10_10_2_node-failure-leaf	250	17:31.08	3980	24:08.59	3980	3980, 3996
10_10_2_node-failure-spine	250	16:21.09	10810	23:40.49	10810	10810, 10833
10_10_2_node-failure-tof	250	16:44.71	34200	23:44.88	34200	34527, 34788
10_10_2_node-recovery-leaf	250	17:46.35	6000	25:01.38	6000	6000, 7976
10_10_2_node-recovery-spine	250	16:41.62	15940	24:22.66	15940	15661, 19401
10_10_2_node-recovery-tof	250	17:13.10	40040	24:49.72	40040	41188, 41667

Table A1. Cont.

Scenario	Number of nodes	Execution time w/o zebra	PDU ns-3 w/o zebra	Execution time w zebra	PDU ns-3 w zebra	PDU Sibyl
10_10_5_link-failure-leaf-spine	100	6:22.98	660	8:42.60	660	660
10_10_5_link-failure-spine-tof	100	6:14.27	760	8:32.91	760	760, 820
10_10_5_link-recovery-leaf-spine	100	6:52.48	781	9:46.89	781	701, 781
10_10_5_link-recovery-spine-tof	100	6:52.53	872	10:36.91	872	811, 882
10_10_5_node-failure-leaf	100	6:43.43	1580	8:57.90	1580	1580
10_10_5_node-failure-spine	100	6:41.49	5410	8:41.55	5410	5410
10_10_5_node-failure-tof	100	6:34.09	11400	8:57.93	11400	11544, 11550
10_10_5_node-recovery-leaf	100	6:31.91	2400	9:57.73	2400	3180, 2790
10_10_5_node-recovery-spine	100	6:39.00	7540	9:25.65	7540	7510, 7490
10_10_5_node-recovery-tof	100	6:38.31	13640	9:25.54	13640	14798, 15178
10_10_10_link-failure-leaf-spine	50	3:09.21	480	4:35.08	480	480
10_10_10_link-failure-spine-tof	50	3:14.54	380	4:21.50	380	380
10_10_10_link-recovery-leaf-spine	50	3:31.83	541	5:10.14	541	501, 541
10_10_10_link-recovery-spine-tof	50	3:15.08	432	4:37.93	432	411, 442
10_10_10_node-failure-leaf	50	3:13.24	780	4:22.19	780	780
10_10_10_node-failure-spine	50	3:11.36	3610	4:14.41	3610	3610
10_10_10_node-failure-tof	50	3:12.86	3800	4:25.57	3800	3800
10_10_10_node-recovery-leaf	50	3:11.31	1200	4:31.73	1200	1580, 1582
10_10_10_node-recovery-spine	50	3:02.23	4740	5:03.15	4740	4670, 4710
10_10_10_node-recovery-tof	50	3:08.48	4840	4:46.55	4840	4840, 5540
12_12_1_link-failure-leaf-spine	720	1:37:58	7464	not enough vRAM	-	-
12_12_1_link-failure-spine-tof	720	1:19:03	12995	not enough vRAM	-	-
12_12_1_link-recovery-leaf-spine	720	1:28:55	8329	not enough vRAM	-	-
12_12_1_link-recovery-spine-tof	720	1:24:22	14102	not enough vRAM	-	-
12_12_1_node-failure-leaf	720	1:22:40	17244	not enough vRAM	-	13800
12_12_1_node-failure-spine	720	1:19:23	78456	not enough vRAM	-	-
12_12_1_node-failure-tof	720	1:19:27	152352	not enough vRAM	-	-
12_12_1_node-recovery-leaf	720	1:20:34	20736	not enough vRAM	-	-
12_12_1_node-recovery-spine	720	1:20:01	104880	not enough vRAM	-	-
12_12_1_node-recovery-tof	720	1:21:15	172848	not enough vRAM	-	-

Table A1. Cont.

Scenario	Number of nodes	Execution time w/o zebra	PDU ns-3 w/o zebra	Execution time w zebra	PDU ns-3 w zebra	PDU Sibyl
12_12_3_link-failure-leaf-spine	240	19:05.59	1488	25:35.76	1488	1488
12_12_3_link-failure-spine-tof	240	19:54.03	2208	25:56.47	2208	2208
12_12_3_link-recovery-leaf-spine	240	21:26.32	1777	29:01.01	1777	1585, 1777
12_12_3_link-recovery-spine-tof	240	20:56.25	2486	28:28.63	2486	2317, 2498
12_12_3_node-failure-leaf	240	19:53.39	5724	28:19.25	4584	4580, 4584, 4628
12_12_3_node-failure-spine	240	19:28.09	15852	26:19.43	15852	15929, 16002
12_12_3_node-failure-tof	240	19:12.64	46368	26:11.43	46368	47227, 57292
12_12_3_node-recovery-leaf	240	19:48.84	6912	26:56.88	6912	8052, 11514
12_12_3_node-recovery-spine	240	19:21.35	21792	28:02.68	21792	21797, 23710, 29768
12_12_3_node-recovery-tof	240	19:31.33	53040	26:09.66	53040	53926, 54024
12_12_4_link-failure-leaf-spine	180	14:05.84	1224	18:58.75	1224	1224
12_12_4_link-failure-spine-tof	180	13:44.66	1656	18:52.29	1656	1656, 1680
12_12_4_link-recovery-leaf-spine	180	14:55.32	1441	21:11.28	1441	1297, 1441
12_12_4_link-recovery-spine-tof	180	15:11.86	1862	21:08.28	1862	1741, 1874
12_12_4_node-failure-leaf	180	14:28.07	3852	19:42.57	3432	3430, 3432
12_12_4_node-failure-spine	180	14:28.08	12684	19:42.68	12684	12763, 12775
12_12_4_node-failure-tof	180	13:56.64	33120	19:12.50	33120	33930, 34800
12_12_4_node-recovery-leaf	180	14:16.45	5184	19:31.86	5184	5184
12_12_4_node-recovery-spine	180	13:50.67	17184	20:41.04	17184	-
12_12_4_node-recovery-tof	180	14:33.37	38064	19:19.03	38064	38979, 39312
12_12_6_link-failure-leaf-spine	120	9:39.02	960	12:19.99	960	960, 983
12_12_6_link-failure-spine-tof	120	8:50.37	1104	12:47.93	1104	1104
12_12_6_link-recovery-leaf-spine	120	10:03.56	1105	14:16.49	1105	1009, 1105
12_12_6_link-recovery-spine-tof	120	10:04.93	1238	13:47.75	1238	1165, 1250
12_12_6_node-failure-leaf	120	9:44.62	2700	12:41.80	2280	2278, 2280, 2292
12_12_6_node-failure-spine	120	9:32.30	9516	12:41.86	9516	9516, 9589
12_12_6_node-failure-tof	120	9:35.47	19872	12:30.12	19872	20580, 20712
12_12_6_node-recovery-leaf	120	9:15.39	3456	12:37.12	3456	3456, 5146
12_12_6_node-recovery-spine	120	9:27.34	12576	13:25.55	12576	12514, 15936
12_12_6_node-recovery-tof	120	9:09.30	23088	12:53.84	23088	23352, 23625

Table A1. Cont.

Scenario	Number of nodes	Execution time w/o zebra	PDU ns-3 w/o zebra	Execution time w zebra	PDU ns-3 w zebra	PDU Sibyl
12_12_12_link-failure-leaf-spine	60	4:14.67	696	5:56.28	696	696
12_12_12_link-failure-spine-tof	60	4:04.78	552	6:10.82	552	552
12_12_12_link-recovery-leaf-spine	60	4:41.71	769	6:56.05	769	721, 769
12_12_12_link-recovery-spine-tof	60	4:38.93	614	6:57.11	614	589, 626
12_12_12_node-failure-leaf	60	4:25.29	1548	6:18.81	1128	1128
12_12_12_node-failure-spine	60	4:12.53	6348	6:08.16	6348	6348
12_12_12_node-failure-tof	60	4:11.17	6624	6:25.72	6624	6624
12_12_12_node-recovery-leaf	60	4:22.37	1728	6:15.41	1728	1728
12_12_12_node-recovery-spine	60	4:24.15	7968	6:31.74	7968	8162, 9469
12_12_12_node-recovery-tof	60	4:23.12	8112	6:25.38	8112	8112, 8676
14_14_1_link-failure-leaf-spine	980	2:42:23	11732	not enough vRAM	-	-
14_14_1_link-failure-spine-tof	980	2:32:38	20817	not enough vRAM	-	-
14_14_1_link-recovery-leaf-spine	980	2:40:38	12909	not enough vRAM	-	-
14_14_1_link-recovery-spine-tof	980	2:39:26	22332	not enough vRAM	-	-
14_14_1_node-failure-leaf	980	2:41:23	27398	not enough vRAM	-	21924
14_14_1_node-failure-spine	980	2:40:11	146384	not enough vRAM	-	-
14_14_1_node-failure-tof	980	2:32:28	285768	not enough vRAM	-	-
14_14_1_node-recovery-leaf	980	2:40:42	32928	not enough vRAM	-	-
14_14_1_node-recovery-spine	980	2:42:50	188608	not enough vRAM	-	-
14_14_1_node-recovery-tof	980	2:36:31	318360	not enough vRAM	-	-
14_14_2_link-failure-leaf-spine	490	54:43.59	3136	59:22.85	3136	-
14_14_2_link-failure-spine-tof	490	52:49.34	5292	58:15.27	5292	-
14_14_2_link-recovery-leaf-spine	490	56:43.90	3725	1:02:30	3725	-
14_14_2_link-recovery-spine-tof	490	56:29.84	5882	1:00:44	5882	-
14_14_2_node-failure-leaf	490	54:45.50	14070	1:01:33	14070	10948
14_14_2_node-failure-spine	490	52:58.71	40782	57:15.31	40782	-
14_14_2_node-failure-tof	490	53:31.63	137592	57:16.83	137592	-
14_14_2_node-recovery-leaf	490	53:24.08	16464	57:33.53	16464	-
14_14_2_node-recovery-spine	490	53:23.65	54740	59:57.35	54740	-
14_14_2_node-recovery-tof	490	54:33.18	153720	58:34.59	153720	-

Table A1. Cont.

Scenario	Number of nodes	Execution time w/o zebra	PDU ns-3 w/o zebra	Execution time w zebra	PDU ns-3 w zebra	PDU Sibyl
14_14_7_link-failure-leaf-spine	140	12:01.43	1316	19:57.69	1316	1336
14_14_7_link-failure-spine-tof	140	11:36.53	1512	18:12.65	1512	1512, 1635
14_14_7_link-recovery-leaf-spine	140	12:29.71	1485	21:14.82	1485	1373, 1485
14_14_7_link-recovery-spine-tof	140	12:07.47	1668	20:51.26	1668	1583, 1682
14_14_7_node-failure-leaf	140	12:13.42	3878	19:14.01	3108	3108, 3150
14_14_7_node-failure-spine	140	12:02.30	15302	19:24.98	15302	15374
14_14_7_node-failure-tof	140	11:36.70	31752	20:30.41	31752	32988, 32990
14_14_7_node-recovery-leaf	140	11:37.59	4704	20:36.14	4704	5470, 7014
14_14_7_node-recovery-spine	140	11:53.92	19460	19:59.21	19460	21484, 24662
14_14_7_node-recovery-tof	140	11:45.48	36120	19:06.72	36120	37967, 39645
14_14_14_link-failure-leaf-spine	70	5:20.93	952	9:06.52	952	952
14_14_14_link-failure-spine-tof	70	5:16.32	756	9:00.30	756	756
14_14_14_link-recovery-leaf-spine	70	5:50.10	1037	10:22.86	1037	981, 1037
14_14_14_link-recovery-spine-tof	70	5:44.78	828	10:41.18	828	799, 842
14_14_14_node-failure-leaf	70	5:30.51	2114	9:50.42	1540	1540, 1552, 1554
14_14_14_node-failure-spine	70	5:39.92	10206	10:21.86	10206	10206, 10308, 10350
14_14_14_node-failure-tof	70	5:32.21	10584	9:21.20	10584	10570, 10584
14_14_14_node-recovery-leaf	70	5:19.37	2352	10:15.21	2352	2730
14_14_14_node-recovery-spine	70	5:32.34	12404	10:15.31	12404	12580, 13024
14_14_14_node-recovery-tof	70	5:50.77	12600	9:44.88	12600	14406, 14574
15_15_1_link-failure-leaf-spine	1125	6:49:52	14370	not enough vRAM	-	-
15_15_1_link-failure-spine-tof	1125	5:10:04	25694	not enough vRAM	-	-
15_15_1_link-recovery-leaf-spine	1125	7:16:43	15721	not enough vRAM	-	-
15_15_1_link-recovery-spine-tof	1125	7:38:04	27437	not enough vRAM	-	-
15_15_1_node-failure-leaf	1125	7:21:43	33705	not enough vRAM	-	-
15_15_1_node-failure-spine	1125	6:57:28	193470	not enough vRAM	-	-
15_15_1_node-failure-tof	1125	7:28:52	378450	not enough vRAM	-	-
15_15_1_node-recovery-leaf	1125	7:14:26	40500	not enough vRAM	-	-
15_15_1_node-recovery-spine	1125	7:06:44	245535	not enough vRAM	-	-
15_15_1_node-recovery-tof	1125	7:06:41	418560	not enough vRAM	-	-

Table A1. Cont.

Scenario	Number of nodes	Execution time w/o zebra	PDU ns-3 w/o zebra	Execution time w zebra	PDU ns-3 w zebra	PDU Sibyl
16_16_1_link-failure-leaf-spine	1280	not enough vRAM		not enough vRAM		-
16_16_1_link-failure-spine-tof	1280	not enough vRAM		not enough vRAM		-
16_16_1_link-recovery-leaf-spine	1280	not enough vRAM		not enough vRAM		-
16_16_1_link-recovery-spine-tof	1280	not enough vRAM		not enough vRAM		-
16_16_1_node-failure-leaf	1280	not enough vRAM		not enough vRAM		32736
16_16_1_node-failure-spine	1280	not enough vRAM		not enough vRAM		-
16_16_1_node-failure-tof	1280	not enough vRAM		not enough vRAM		-
16_16_1_node-recovery-leaf	1280	not enough vRAM		not enough vRAM		-
16_16_1_node-recovery-spine	1280	not enough vRAM		not enough vRAM		-
16_16_1_node-recovery-tof	1280	not enough vRAM		not enough vRAM		-
16_16_8_link-failure-leaf-spine	160	16:09.07	1728	27:12.65	1728	1728
16_16_8_link-failure-spine-tof	160	15:58.80	1984	23:39.03	1984	1984
16_16_8_link-recovery-leaf-spine	160	17:09.07	1921	27:48.62	1921	1793, 1991
16_16_8_link-recovery-spine-tof	160	16:58.23	2162	26:25.27	2162	2065, 2178
16_16_8_node-failure-leaf	160	16:24.14	5584	24:45.55	4064	4062, 4064
16_16_8_node-failure-spine	160	16:31.02	23056	23:16.42	23056	23262
16_16_8_node-failure-tof	160	16:35.37	47616	23:58.30	47616	48543, 48807
16_16_8_node-recovery-leaf	160	15:59.40	6144	24:17.76	6144	8230, 10174
16_16_8_node-recovery-spine	160	16:20.55	28480	25:01.35	28480	29729, 30013
16_16_8_node-recovery-tof	160	16:54.93	53312	24:16.86	53312	54467, 56962
16_16_16_link-failure-leaf-spine	80	7:09.04	1248	12:02.83	1248	1248, 1310
16_16_16_link-failure-spine-tof	80	7:16.87	992	11:04.84	992	992
16_16_16_link-recovery-leaf-spine	80	7:42.15	1345	13:31.50	1345	1281, 1345
16_16_16_link-recovery-spine-tof	80	7:44.84	1074	13:39.60	1074	1041, 1090
16_16_16_node-failure-leaf	80	7:32.59	2768	12:06.73	2016	2016
16_16_16_node-failure-spine	80	7:27.18	15376	11:50.89	15376	15582, 15596, 15599
16_16_16_node-failure-tof	80	7:38.51	15872	12:19.22	15872	15376, 15840, 15872
16_16_16_node-recovery-leaf	80	7:29.09	3072	12:13.98	3072	3070, 4064
16_16_16_node-recovery-spine	80	7:21.55	18240	13:44.53	18240	18016, 19074
16_16_16_node-recovery-tof	80	7:21.64	18496	12:05.75	18496	19328, 19776

References

1. Cisco. *Cisco Global Cloud Index: Forecast and Methodology, 2016–2021*; White Paper; 2018; Cisco; USA.
2. Clos, C. A study of non-blocking switching networks. *Bell Syst. Tech. J.* **1953**, *32*, 406–424. <https://doi.org/10.1002/j.1538-7305.1953.tb01433.x>.
3. Alberro, L.; Castro, A.; Grampin, E. Experimentation Environments for Data Center Routing Protocols: A Comprehensive Review. *Future Internet* **2022**, *14*, 29. <https://doi.org/10.3390/fi14010029>.
4. Bonofiglio, G.; Iovinella, V.; Lospoto, G.; Di Battista, G. Kathará: A container-based framework for implementing network function virtualization and software defined networks. In Proceedings of the NOMS 2018—2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–9. <https://doi.org/10.1109/NOMS.2018.8406267>.
5. Scazzariello, M.; Ariemma, L.; Caiazzi, T. Kathará: A Lightweight Network Emulation System. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–2. <https://doi.org/10.1109/NOMS47738.2020.9110351>.
6. Scazzariello, M.; Ariemma, L.; Battista, G.D.; Patrignani, M. Megalos: A Scalable Architecture for the Virtualization of Network Scenarios. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–7. <https://doi.org/10.1109/NOMS47738.2020.9110288>.
7. Ahrenholz, J. Comparison of CORE network emulation platforms. In Proceedings of the 2010—MILCOM 2010 Military Communications Conference, San Jose, CA, USA, 31 October–3 November 2010; pp. 166–171. <https://doi.org/10.1109/MILCOM.2010.5680218>.
8. Lantz, B.; Heller, B.; McKeown, N. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Monterey, CA, USA, 20–21 October 2010; Association for Computing Machinery: New York, NY, USA, 2010; Hotnets-IX. <https://doi.org/10.1145/1868447.1868466>.
9. Caiazzi, T.; Scazzariello, M.; Alberro, L.; Ariemma, L.; Castro, A.; Grampin, E.; Battista, G.D. Sibyl: A Framework for Evaluating the Implementation of Routing Protocols in Fat-Trees. In Proceedings of the NOMS 2022—2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 25–29 April 2022; pp. 1–7. <https://doi.org/10.1109/NOMS54207.2022.9789876>.
10. Lapukhov, P.; Premji, A.; Mitchell, J. *Use of BGP for Routing in Large-Scale Data Centers*; RFC 7938, RFC Editor; 2016; IETF. Available online: <https://datatracker.ietf.org/doc/rfc7938/> (accessed on 12 October).
11. White, R.; Hegde, S.; Zandi, S. IS-IS Optimal Distributed Flooding for Dense Topologies. Internet-Draft Draft-White-Distoptflood-03, IETF Secretariat, 2020. Available online: <https://datatracker.ietf.org/doc/html/draft-white-distoptflood-03> (accessed on 12 October).
12. Przygienda, T.; Sharma, A.; Thubert, P.; Rijsman, B.; Afanasiev, D.; Head, J. RIFT: Routing in Fat Trees. Internet-Draft Draft-ietf-rift-rift-16, IETF Secretariat, 2022. Available online: <https://datatracker.ietf.org/doc/draft-ietf-rift-rift/> (accessed on 12 October).
13. Aelmans, M.; Vandezande, O.; Rijsman, B.; Head, J.; Graf, C.; Alberro, L.; Mali, H.; Steudler, O. *Day One: Routing in Fat Trees (RIFT)*; Juniper Networks Books: 2020.USA.
14. Quagga. Available online: <https://www.quagga.net/> (accessed on 1 August 2022).
15. Tazaki, H.; Uarbani, F.; Mancini, E.; Lacage, M.; Camara, D.; Turletti, T.; Dabbous, W. Direct code execution: Revisiting library architecture for reproducible network experiments. In Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, 2013; Santa Barbara, CA, 9–12 December 2013. pp. 217–228.
16. ns-3 Network Simulator. Available online: <https://www.nsnam.org> (accessed on 30 September 2022).
17. ns-3 Direct Code Execution. Available online: <https://www.nsnam.org/about/projects/direct-code-execution> (accessed on 30 September 2022).
18. Caiazzi, T.; Scazzariello, M.; Ariemma, L. VFTGen: A Tool to Perform Experiments in Virtual Fat Tree Topologies. In Proceedings of the IM 2021—2021 IFIP/IEEE International Symposium on Integrated Network Management, Virtual, 17–21 May 2021.
19. Sibyl results. Available online: <https://gitlab.com/uniroma3/compunet/networks/sibyl-framework/sibyl-results> (accessed on 30 September 2022).
20. Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *2014*, 2.
21. Kubernetes. Kubernetes. 2021. Available online: <https://kubernetes.io/> (accessed on 30 September 2022).
22. Azpiroz, S.Y.; Velázquez, F. FRR ns-3 DCE. 2021. Available online: <https://gitlab.com/fing-mina/datacenters/frr-ns3> (accessed on 30 September 2022).
23. ns-3 Manual. Available online: <https://www.nsnam.org/docs/release/3.34/manual/singlehtml/index.html> (accessed on 30 September 2022).
24. Kaashoek, M.F.; Engler, D.R.; Ganger, G.R.; Briceño, H.M.; Hunt, R.; Mazières, D.; Pinckney, T.; Grimm, R.; Jannotti, J.; Mackenzie, K. Application Performance and Flexibility on Exokernel Systems. In Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, Saint Malo, France, 5–8 October 1997; Association for Computing Machinery: New York, NY, USA, 1997; SOSP '97; pp. 52–65. <https://doi.org/10.1145/268998.266644>.
25. ns-3 Direct Code Execution (DCE) Documentation. 2021. Available online: <https://ns-3-dce.readthedocs.io/en/latest/intro.html> (accessed on 30 September 2022).
26. White, R.; Zandi, S. IS-IS Support for Openfabric. Internet-Draft Draft-White-Openfabric-07, IETF Secretariat, 2018. Available online: <https://datatracker.ietf.org/doc/html/draft-white-openfabric-07> (accessed on 12 October).

27. DCE Quagga. Available online: <https://www.nsnam.org/docs/dce/manual-quagga/html/getting-started.html> (accessed on 30 September 2022).
28. Fix bug in dce vasprintf. Available online: <https://github.com/direct-code-execution/ns-3-dce/pull/132> (accessed on 30 September 2022).
29. Fix bug in dce internalClosedir. Available online: <https://github.com/direct-code-execution/ns-3-dce/pull/133> (accessed on 30 September 2022).
30. Azpiroz, S.Y.; Velázquez, F. FRR Compilation and Installation Script for ns-3 DCE. 2021. Available online: <https://gitlab.fing.edu.uy/proyecto-2021/scripts/-/blob/master/04-install-frr-SUDO> (accessed on 30 September 2022).
31. Free Range Routing. Available online: <https://frrouting.org> (accessed on 30 September 2022).
32. Kathara-Labs. Available online: <https://github.com/KatharaFramework/Kathara-Labs> (accessed on 30 September 2022).
33. Medhi, D.; Ramasamy, K. *Network Routing, Second Edition: Algorithms, Protocols, and Architectures*, 2nd ed.; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2017.