4.5 Example Deployment

This example shows how to create an experiment model, build a testbed and run experiments. Next to this step by step guide, the repository [TR] contains README files as a guide, which are also shown in the attached directory tree 8.2. The required packages for this example are the following:

- qemu-utils
- python3
- python3-pip
- pyangbind (pip3 install pyangbind)
- networkx (pip3 install networx)
- pygraphviz and utilities (sudo apt install graphviz libgraphviz-dev pkg-config && pip3 install pygraphviz)
- Linux kernel version 4.9.0-13-amd64 with Xen [TLF] hypervisor extension for local topology
- Virtualbox [OC] version 5.2.42 or later for encapsulated topologies

Unless a dedicated machine for executing this example is available, which can be downgraded to the required kernel version and provisioned with the Xen hypervisor [TLF], it is recommended to build an encapsulated topology with VirtualBox [OC]. If you choose to build a local topology, be informed that the directories */xen* and */control* will be created on the local file system and that any contents could be overwritten. References to the project repository and VM images are also required and are part of the work of Tran [Tra22]. They are accessible through the project repository [TR].
To install the necessary dependencies please execute one of the following scripts:

```
$: cd topogen/sdn_virtualbox_installs
$: cd topogen/sdn_xen_installs
```

The first step is the creation of an experiment based on the YANG module. Open a command line terminal in the project repository and execute commands:

```
$: cd topogen/autogen
$: touch sample_topology.py
```

Use any desired text editor and create a model. The below code is an example experiment with four switches and four hosts as a testbed, illustrated by Figure 3.2. We deploy the Host Shadower and Path Load Balancer applications. The code shows the use of a model for a custom application that can be provided by a researcher. The *someApp* application needs to be provided by a researcher, either based on a Openflow REST interface as a Python module but independent of any controller framework, or as a Ryu [Nip] control application. In this example file has to be named *someApp.py* and copied to the directory controller/massive of the repository. In this example we assume that the application does not actually use the generated config file from the model, but it still provides a configuration to declare interest

for hosts pc1 and pc2 as well as switch router1. A definition of a cookie, which should be used for flow entries as well as a declaration of at least one target switch, is mandatory. If you do not want to provide a custom application, comment out the respective lines in the code. Below the application configurations, we add hosts, switches and edges to the topology and draw a diagram.

```
import os
import json
import random_topo
import pyangbind.lib.pybindJSON as pybindJSON
from pyangbind.lib.serialise import pybindJSONDecoder
from sdn_testbed_spec_v2 import sdn_testbed_spec_v2
from constants_generate_spec import TOPOLOGY_DIR

topoId = "sample_topology"
topoFileName = os.path.join(TOPOLOGY_DIR, "".join([topoId, ".json"]))
model = sdn_testbed_spec_v2()

model.testbed._set_autostart("true")
model.testbed.topologyId = topoId

# Host Shadower config
hs = model.testbed.apps.add("hs")
hs.config.cookie = "0x440"
hs.config.targetSwitches.add("router2")
hs.config.targetSwitches.add("router3")
# add(1) is just an initializer and has no further meaning
appAssetsConfig = hs.config.appAssets.add(1)
appAssetsConfig.majorAsset.assetKey = "frontend"
appAssetsConfig.majorAsset.assetValue = "pc2"
appAssetsConfig.minorAssets.assetKey = "backend"
appAssetsConfig.minorAssets.assetValue = "pc3"

# Path Load Balancer config
plb = model.testbed.apps.add("plb")
plb.config.cookie = "0x300"
bw = plb.config.appInvariants.add("bw_time")
bw.intValue = 5
ts = plb.config.targetSwitches.add("router1")
bw_thres = ts.switchInvariants.add("bw_threshold")
bw_thres.intValue = 10

# configuration for custom app
# which does not use config file but
# provides config for automated results
someApp = model.testbed.apps.add("someApp")
someApp.config.cookie = "0x200"
```

```
someApp.config.targetSwitches.add("router4")
appAssetsConfig = someApp.config.appAssets.add(1)
appAssetsConfig.minorAssets.assetKey = "hosts"
appAssetsConfig.minorAssets.assetItems = ["pc1","pc2"]
appAssetsConfig = someApp.config.appAssets.add(2)
appAssetsConfig.minorAssets.assetKey = "switch"
appAssetsConfig.minorAssets.assetValue = "router1"

# set traffic profile and types
model.testbed.trafficTypes = ["udp", "tcp"]
model.testbed.trafficProfiles = ["cbr", "vbr", "bursty"]

# add switches and hosts, set host 1 as traffic source
for i in range(1,5):
  model.testbed.switches.add("router{}".format(i))
  pc = model.testbed.hosts.add("pc{}".format(i))
  if i == 1:
    pc._set_source("true")

# set the edges between nodes
# hosts need to be connected to one switch
# switches are connected to a switch or host
router_edges = [["router1","router2"],
    ["router1","router3"],["router2","router3"],
    ["router2","router4"],["router3","router4"]]

# draw the switches, hosts not supported yet
agraph = random_topo.create_agraph_fromedges(router_edges)
random_topo.draw_switches(agraph, topoId)

host_edges = [["router1","pc1"],["router4","pc2"],
    ["router4","pc3"],["router4","pc4"]]
edges = host_edges + router_edges

for i in range(len(edges)):
  edge = model.testbed.edges.add(i)
  edge.nodes = edges[i]

# write yang model to json in ietf format
output = pybindJSON.dumps(model, mode="ietf")
print(output, file=open(topoFileName, "w"))
```

Listing 4.2: Sample configuration for an experiment on Software Defined Network conflicts, including specifications for the network topology shown in Figure 3.2. Source hosts generate traffic, all other endpoints start mock servers for services. The current implementation supports automated generation of network diagrams for switch nodes.

To generate the experiment JSON file, run the following command within the directory topogen/autogen of the repository.

```
$: python3 sample_topology.py
```

The directory topogen/topology contains experiment models for all topologies listed in Section 3 with various app configurations and can be used for generating a JSON model instance for this example. Adjust the constants in the file *constants_generate_spec.py* to the local file system, provide absolute paths to template images and a path to the optional SSH config and cryptographic key-pair. To generate the specification files for building the topology and any app configuration files, execute the following command within the directory topogen/autogen:

```
$: python3 generate_sdn_spec.py path/to/sample_topology.json
```

This produces specification files in the directory topogen/specs and will show the absolute path on the command line. Be sure to check for the correct path to the generated specification files. Now move to the directory topogen/scripts and execute one of the following command to build an encapsulated topology:

```
$: sudo deploy_sdn_conflicts_experiment.sh ../specs/sample_topology
```

You will be prompted for the amount of processors that can be allocated to the VM and a number to create a local port forwarding for SSH access to the VM. The required SSH command will be printed on the command line.

For deploying a Xen [TLF] topology locally, execute the alternative script:

```
$: sudo build_sdn_conflicts_experiment.sh ../specs/sample_topology
```

Root privileges are required for both scripts,since the implementation mounts images to the local file system.

For a successful deployment within a Virtualbox [OC] VM, it can now be accessed. Since the experiment model contains the autostart attribute, no further steps are required. As soon as a tmux [Mar] session is created, the experiment run has started. Switch to the root user and access the the VM via SSH with the following command if your provided port was e.g. 2226:

```
#: sudo -s && ssh -p 2226 localhost
```

Now check if a tmux [Mar] session has been created and attach to it.

```
#: tmux ls
#: tmux attach -t ID
```

For a locally deployed Xen [TLF] topology check, for any new tmux [Mar] session with the above commands and attach to it. If you provided a SSH config, any results will be uploaded to the target machine within the directory *sdn_results*. If you want to start more experiment runs, create more experiment models, and insert the JSON files into the encapsulated VM if necessary. From there move to the directory /control/automating_experiment and execute the following command:

```
#: bash iterate_parameter_space.bash path_to_json_files
```

This will only work for models that fit the generated testbed. The testbed is not recreated, but the application configurations are based on nodes in the testbed and will only be valid for these nodes. For every JSON model instance the script *app_config_generator.py* will generate app configuration files and any required global settings for an experiment run. This process is used for experiment building, as well as experiment deployment and the script will be imported automatically where needed. The script can also be used to automatically generate configuration files from an experiment model for a manual execution of the Ryu [Nip] controller and control applications. The experiment results are saved to the directory */control/automating_experiment/dataset* or on the configured remote machine under */home-/user/sdn_results*.

When building a new testbed, be sure to tear down an established topology in order to free up needed resources and avoid deploying any remnant Xen [TLF] domains in case of a local deployment. This also applies for removing testbeds after finishing experiments. For an encapsulated topology, simply stop and delete the wrapper VM. To determine the name of a wrapper VM, try either of the following commands.

```
#: vboxmanage list vms
#: vboxmanage list runningvms
```

Be sure to collect any generated results from */control/automating_experiment/dataset* if you did not supply a valid SSH configuration. To stop and delete the VM, execute both commands:

```
#: vboxmanage controlvm sample_topology-1633001718 poweroff
#: vboxmanage unregistervm sample_topology-1633001718 --delete
```

In case of a local topology, the control folder needs to be deleted and the created Xen [TLF] domains need to be destroyed and deleted:

```
#: rm -r control
#: /xen/rnp_vms stop
```

Check if all Xen [TLF] domains have been removed:

```
#: xen list
```

If only the domain with id 0 shows in the list, all files can be deleted:

```
#: rm -r /xen
```

The presented approach promotes sharing of topology designs between researchers. It integrates a universal set of tools that maintain close control over technical details, while automating mundane tasks that result in configuration errors and increased workload for topology setup.