Automatyzacja przenoszenia konfiguracji pomiędzy środowiskiem symulowanym oraz sprzętowym

Michał Skorek, Szymon Stępień, Arkadiusz Wołk

1 Środowiska wirtualizacji

W ramach projektu przeprowadzono przegląd różnych narzędzi umożliwiających wirtualizacje sieci w celu dobrania odpowiedniego do przenoszenia konfiguracji. Sekcja przedstawia krótki opis uwzględnianych narzędzi, ich mocne strony oraz braki.

1.1 Kathara

Opcje:

- Obsługa kontenerów, wraz z predefiniowanymi obrazami Dockera m.in. dla:
  - Quagga - podstawowy routing, m.in. OSPF i BGP, konfigurowalny za pośrednictwem plików testowych, konfiguracja podobna do Cisco IOS;
  - FRRouting - bardziej rozbudowana Quagga;
  - P4 - bmv2 switch, p4runtime;
  - OpenVSwitch - wraz z kontrolerem Ryu.
- Wygodne CLI i konfiguracja topologii (tzw. Labów) za pomocą systemu plików, kontenery "przypina się" do domen kolizyjnych (Linux bridge), dla każdego można zdefiniować plik tekstowy, z którego komendy zostaną wykonane po uruchomieniu kontenera (pozwala to np. na konfiguracje adresacji). Dla kontenerów można utworzyć folder z plikami, które zostaną automatycznie do nich przekopiowane, co pozwala na konfigurację niektórych programów systemu Linux za pomocą plików konfiguracyjnych, np. routingu w Quagga.
- API do Pythona pozwalające na tworzenie topologii z kodu.

Zalety:

- Bardzo dużo przykładów topologii (w konfiguracji z systemu plików), dobrze udokumentowanych, wraz z bardziej zaawansowanej przypadkami, np. w pełni skonfigurowane fat tree. W szczególności warto zobaczyć repozytorium Kathara-Labs[3] zawierające działające przykłady wraz z slajdami opisującymi jak to działa.
- Można wykorzystywać dowolne inne kontenery Dockera, w szczególności bazowy, który reprezentuje zwykłego hosta, wtedy mamy sytuację bardzo zbliżoną do Minineta.
- Dzięki wirtualizacji z wykorzystaniem kontenerów (a nie maszyn wirtualnych) możliwe jest uruchamianie bardzo dużych sieci (chociaż w przypadku tego projektu nie powinno to być konieczne).
- Dobra dokumentacja CLI i samego kodu, dużo komentarzy.

Wady:

- Narzędzie zostało bardziej dostosowane do budowy topologii za pomocą podejścia z systemem plików niż za pomocą API Pythonowego. Jest całkiem niezła dokumentacja tego API jednak ciężko znaleźć jakiekolwiek przykłady konfiguracji topologii wykorzystujące API. Model danych jest mocno ograniczony i brakuje jakichkolwiek reprezentacji (np. klas Pythonowych) dla protokołów sieciowych, wszystko trzeba robić w formacie tekstowym.

- Switch nie jest osobnym urządzeniem i nie jest modelowany przez Kathare. Wykorzystywane jest pojęcie domen kolizyjnych (pod spodem implementowanych jako Linux bridge), w związku z tym wszelkie operacje związane z konfiguracją switchy należałoby zrobić samodzielnie wykorzystując API Linuxa. Można również spróbować wykorzystać kontener zbliżony do switcha (np. P4 BMv2).
- Ograniczone wykorzystanie pewnych protokołów, przykładowo GRE trzeba byłoby dorobić samodzielnie, jednak z uwagi na wykorzystanie sieciowych przestrzeni nazw systemu Linux przez Kathare nie powinno być to bardzo problematyczne.

1.2 Mininet oraz Containernet

Narzędzia te są o tyle podobne do Kathary, że również bazują na systemie przestrzeni nazw w Linux'ie i łączą je za pomocą wirtualnych interfejsów i bridge'ów. Są one jednak bardziej dostosowane do wirtualizacji sieci SDN (przykładowo są klasy reprezentujące kontrolery SDN'owe oraz wbudowana obsługa OpenVSwitch'a), Mininet kompletnie nie nadaje się do przenoszenia konfiguracji z/do klasycznych sieci.

W przypadku Containernet będącego forkiem Minineta z obsługą kontenerów Dockera jest już trochę lepiej, ponieważ jako urządzenia końcowe można wykorzystać chociażby te same kontenery, które wykorzystuje Kathara (czyli np. Quagga).

Containernet ma niezłe API do Pythona, z wieloma przykładami wykorzystania, jednak ponownie konfiguracja jest dokonywana tekstowo. W tym przypadku jest to jeszcze mniej wygodne niż Kathara, bo cała konfiguracja jest dokonywana z Pythona, a więc nasze narzędzie do przenoszenia konfiguracji musiałoby prawdopodobnie parsować oraz generować kod Pythona, a nie pliki tekstowe.

1.3 GNS-3

Opcje:

- Narzędzie bardziej ogólne niż Kathara i Containernet, pozwala na budowanie topologii w których hosty/routery/switche są maszynami wirtualnymi.
- Hostem może być również kontener Docker'a, więc potencjalnie te wszystkie opcje wyżej również mogłyby działać.
- GUI oraz REST-API do tworzenia topologii, istnieją też bindingi tego API do Pythona[8].

Zalety:

- Obszerna dokumentacja i potencjalnie większa swoboda, narzędzie dosyć popularne.
- Można (nie w pełni legalnie) wykorzystać w topologii zwykłe urządzenia Cisco z ich IOS'em, zarówno routery jak i switche.
- Teoretycznie jak już dałoby się zrobić topologie z poziomu kodu całe przenoszenie byłoby ograniczone do wywołania komendy *show running-config* na urządzeniu fizycznym i wgraniu jej wyjścia do urządzenia wirtualnego, analogicznie w drugą stronę.

Wady:

- Mało przykładów API do tworzenia topologii, po krótkich testach było to dosyć problematyczne i ograniczone, raczej narzędzie bardziej dostosowane do używania z GUI.
- Dużo bardziej skomplikowane niż wcześniejsze dwa.
- Problemy ze znalezieniem obrazów Cisco, szczególnie nowszych. Na pewnych stronach bazujących na protokole BitTorrent można znaleźć starsze obrazy[1], jednak przykładowo żaden z nich nie obsługiwał netconf'a. Użyteczność GNS-3 bez obrazów jest bardzo ograniczona.

[1] podobno, autorzy tego raportu oczywiście nie praktykowali takich rzeczy

1.4 Podsumowanie

Uwzględniając powyższe zdecydowaliśmy się na wykorzystanie Kathary, rozważając ilość pracy implementacyjnej, szczególnie związanej z parsowaniem i serializacją danych, ostatecznie mogła to nie być najlepsza decyzja i GNS-3 byłby lepszym wyborem (pod warunkiem znalezienia odpowiednich obrazów).

2 Implementacja rozwiązania

Sekcja zawiera opis rozwiązania[1], przedstawia krótko model danych, uzasadnia decyzje implementacyjne, wymienia braki i opisuje potencjalny kierunek dalszego rozwoju projektu.

2.1 Idea rozwiązania

Środowisko wirtualne stanowi Kathara, jak wspomniano model danych nie obejmuje ani switchy ani protokołów, jedynie abstrakcyjne *maszyny* będące kontenerami Dockera, które można łączyć do domen kolizyjnych oraz konfigurować w sposób tekstowy za pomocą reprezentacji systemu plików.

Wprowadzamy ogólny model danych, niezależny od samej Kathary ani urządzeń fizycznych. Obejmuje on urządzenia (Router, Switch), interfejsy, oraz protokoły (np. RIP), a także inne konstrukcje sieciowe (np. routing statyczny). Program buduje topologię sieci, tzn. graf nieskierowany, gdzie węzłami są urządzenia, a krawędzie reprezentują fizyczną łączność w warstwie 2. Urządzenia posiadają referencje do swoich interfejsów sieciowych oraz protokołów, które zostały/mają zostać na nich skonfigurowane. Program ma interfejs CLI, a jego działanie wymaga interakcji użytkownika.

Przenoszenie konfiguracji z środowiska wirtualnego na fizyczne przebiega następująco, zakładamy, że urządzenia fizyczne nie są w żadnym stopniu skonfigurowane:

1. Na podstawie konfiguracji z sytemu plików Kathary program buduje topologię sieci, tworzy urządzenia, łączy je w graf, a następnie parsuje wszelkie pliki konfiguracyjne dotyczące m.in. adresacji oraz protokołów sieciowych. Przyjmujemy, że jeśli w wirtualnej domenie kolizyjnej są 2 urządzenia jest to bezpośrednie połączenie, w przeciwnym przypadku jest to switch.

2. Program przechodzi kolejno po wszystkich urządzeniach w grafie i prosi użytkownika o podłączenie portu konsolowego, kolejno, dla każdego urządzenia.

3. Po potwierdzeniu przez użytkownika program konfiguruje ssh, netconf, CDP oraz interfejs ethernet, wykorzystując który można połączyć się z urządzeniem.

4. Po wstępnej konfiguracji wszystkich urządzeniem program prosi kolejno o ich podłączenie poprzez skonfigurowany interfejs ethernet (żeby nie przepinać wielokrotnie urządzeń sugerujemy podłączenie wszystkich do wspólnego switcha na czas konfiguracji).

5. Po potwierdzeniu przez użytkownika program konfiguruje wszelkie protokoły sieciowe, które były skonfigurowane w wirtualnej sieci z wykorzystaniem protokołu netconf.

6. Program informuje użytkownika, że konfiguracja została zakończona i wyświetla rysunek topologii sieci.

Przenoszenie konfiguracji z topologii fizycznej na środowisko wirtualne przebiega następująco:

1. Użytkownik uruchamia program podając ile routerów i ile switchy znajduje się w topologii.
2. Program dla każdego z routerów i switchy prosi użytkownika o podłączenie portu konsolowego.
3. Po potwierdzeniu przez użytkownika program konfiguruje ssh, netconf, CDP oraz interfejs ethernet, wykorzystując który można połączyć się z urządzeniem, ponadto program zapisuje wyjście komendy *show running-config*.
4. Program przetwarza wyjście komendy *show running-config* zapisując tzw. wskazówki dotyczące rzeczy, które zostały skonfigurowane na urządzeniu, w celu późniejszego odczytania ich dokładnej konfiguracji.
5. Po wstępnej konfiguracji wszystkich urządzeniem program prosi kolejno o ich podłączenie poprzez skonfigurowany interfejs ethernet.
6. Po potwierdzeniu przez użytkownika program czyta konfiguracje protokołów sieciowych bazując na wspomnianych wskazówkach z wykorzystaniem protokołu netconf. Program parsuje wyniki w formacie XML (na podstawie standardowych modeli yang) i tworzy odpowiednie obiekty zgodne z zdefiniowanym modelem danych.
7. Program buduje graf sieci z wykorzystaniem protokołu CDP, zakładamy, że w tym momencie CDP zdążyło zbiegnąć.
8. Program dokonuje serializacji modelu do systemu plików zgodnego z wymaganiami Kathary
9. Program informuje użytkownika, że konfiguracja została zakończona i wyświetla rysunek topologii sieci.

2.2 Aktualny stan implementacji i braki

Struktura powyższej idei została zaimplementowana w podstawowym zakresie, brakuje jedynie budowy grafu sieci przy przenoszeniu z topologii fizycznej na wirtualną. Z uwagi na bardzo utrudnione możliwości testowania programu w trybie zdalnym wymienimy poniżej rzeczy, które zostały zaimplementowane i przetestowane, rzeczy tylko zaimplementowane, oraz rzeczy, które powinny zostać jeszcze zaimplementowane.

Rzeczy zaimplementowane i przetestowane (w granicach możliwości):

- Parsowanie systemu plików Kathary na nasz model danych, w zakresie:
  - budowanie topologii,
  - adresacja,
  - routing statyczny,
  - RIP,
  - OSPF,
  - PAT, SNAT, DNAT;
- Serializacja naszego modelu na system plików Kathary, w zakresie opisanym wyżej (oprócz RIPa i OSPFa);
- Konfiguracja i czytanie konfiguracji urządzeń poprzez port konsolowy, w zakresie wymienionym w sekcji 2.1;

- CLI umożliwiające uruchamianie programu w trybach przenoszenia konfiguracji w wymaganą stronę oraz jedynie wizualizację wirtualnej topologii, wszelkie komunikaty dla użytkownika dotyczące podpinania urządzeń itp;
- Podstawowa wizualizacja topologii jako grafu.

Rzeczy zaimplementowane i częściowo bądź wcale nieprzetestowane:

- Konfiguracja wyżej wspomnianych protokołów, adresacji i routingu z wykorzystaniem netconfa;
- Odczyt wyżej wspomnianych protokołów, adresacji i routingu z wykorzystaniem netconfa;

Rzeczy do zaimplementowania:

- Budowanie grafu sieci zgodnego z naszym modelem danych na podstawie fizycznej topologii i być może protokołu CDP (należy odczytać dane o interfejsach fizycznych oraz połączeniach i je odpowiednio przypisać);
- Implementacja obsługi OSPF oraz BGP (parsowanie/serializacja z/do modelu Kathary - Quagga, oraz zapis/odczyt z/do urządzeń fizycznych) oraz NAT'a (zapis/odczyt z/do urządzeń fizycznych);
- Potencjalnie obsługa firewalla wykorzystując iptables (sekcja 3);
- Implementacja obsługi protokołu GRE, wykorzystując narzędzia dostępne w systemie Linux;
- Obsługa konfiguracji switchy, wykorzystując narzędzia dostępne w systemie Linux.

Potencjalne problemy:

- z punktu praktycznego ważne może być, że nie wszystkie urządzenia w sali sieciowej obsługują netconfa (z tego co sprawdzaliśmy 2 routery z każdego zestawu i chyba nie wszystkie switche). Być może konieczny byłby jakiś fallback programu do konfiguracji na podstawie *show running-config*, bezpośrednio poprzez port konsolowy (zredukowałoby to też ilość przepinania kabli);
- problem z przypisaniem fizycznych interfejsów przy przenoszeniu konfiguracji z wirtualnej na fizyczną. W sposób automatyczny praktycznie (chyba) nie da się sprawdzić, w które interfejsy można wpiąć kabel (komenda *show ip interface* wypisuje takie interfejsy mimo, że fizycznie router nie pozwala na ich wykorzystanie).

2.3 Model danych

Główną strukturą danych jest węzeł sieci (**Node**), dziedziczy z niej **CiscoNetworkNode**, z którego z kolei dziedziczą **Router** i **Switch**. Z klasy **Node** dziedziczy ponadto **Host** (uwzględniany w modelu Kathary).

```
Node:
  string              name          : nazwa urządzenia w topologii wirtualnej
  map[str, Interface] interfaces    : kluczem jest nazwa interfejsu w topologii
                                      wirtualnej, wartością jest sam interfejs
  map[str, Node]      neighbours    : kluczem jest nazwa interfejsu, do którego
                                      podłączony jest sąsiad urządzenia, wartością
                                      sam sąsiad
  list[StaticRoute]   static_rotues : trasy statyczne urządzenia

CiscoNetworkNode (dziedziczy z Node):
  string          running_conf_data : wyniki komendy show running-config
  Interface       netconf_interface : interfejs do konfiguracji przez netconf
```

Router zawiera dane dotyczące różnych skonfigurowanych protokołów, dla zwięzłości strukturę samych protokołów pominiemy.

```
Router (dziedziczy z CiscoNetworkNode):
  list[Snat]        snat_rules          : lista zaaplikowanych zasad translacji adresu źródłowego
  list[Dnat]        dnat_rules          : lista zaaplikowanych zasad translacji adresu docelowego
  list[OverloadNat] overload_nat_rules  : lista zaaplikowanych zasad translacji adresu źródłowego
                                          z przeciążeniem (PAT)
  RipConfig         rip_config          : informacje konfiguracyjne protokołu RIP
  OSPFConfig        ospf_config         : informacje konfiguracyjne protokołu OSPF
```

Interfejs sieciowy (**Interface**) jest reprezentowany w poniższy sposób:

```
Interface:
  string virtual_name  : nazwa interfejsu w topologii wirtualnej
  string physical_name : nazwa interfejsu w topologii fizycznej
  string ipv4          : adres IP interfejsu (w notacji kropkowanej)
  int    netmask       : maska sieciowa interfejsu (0-32)
  bool   enabled       : informacja czy interfejs jest aktywny
  bool   used          : informacja czy interfejs jest wykorzystywany
```

Graf sieci (**Topology**) jest reprezentowany jako list węzłów. Pozostałe modele powinny być oczywiste po przeczytaniu kodu, dlatego dla zwięzłości je tutaj pomijamy.

2.4 Konfiguracja routingu

W [13] są przedstawione wszystkie obsługiwane protokoły przez Quagga oraz sposób ich konfiguracji przy pomocy plików konfiguracyjnych. Każdemu z protokołów odpowiada osobny plik, w którym przy pomocy komend można uzyskać porządany efekt. Obsługiwane protokoły:

1. RIP
2. RIPng
3. OSPFv2
4. OSPFv3
5. ISIS
6. NHRP
7. BGP

2.4.1 RIP

Na tą chwilę narzędzie obsługuje nie wszystkie komendy do konfiguracji RIP udostępnianie przez Quagga, co także wynika z ograniczeń obsługi RIP przez protokół netconf [14] (nie ma pewności, że jakiekolwiek urządzenie w laboratorium obsługuje to rozszerzenie).
Przy pomocy protokołu netconf możliwe jest skonfigurowanie następujących ustawień:

1. Redystrybucja tras

2. Czasów:
   (a) update
   (b) invalid
   (c) holddown
   (d) flush
3. Interfejsów (na których protokół ma być obsługiwany):
   (a) nazwa interfejsu
   (b) adresy ip sąsiadów

Natomiast nasze narzędzie obsługuje następujące komendy Quagga:

1. version <n> - ustawiienie wersji protokołu RIP
2. network <ifname> - interfejs 'ifname' będzie obsługiwał protokół RIP
3. network a.b.c.d/m - interfejsy, które należą do tej sieci będą obsługiwały protokół RIP
4. neighbor a.b.c.d - dodanie sąsiada
5. redistribute <prot> - włączenie redystrybucji tras z protokołu 'prot'

Jedynie dodawanie interfejsów poprzez specyfikację adresu sieci oraz dodawanie sąsiadów nie jest wprost obsługiwane przez netconf, jednak udało nam się rozwiązać ten problem.

Dodawanie sąsiadów jest możliwe w netconf na poziomie każdego z interfejsów, więc aby poprawnie skonfigurować protokół, decyzja gdzie dodać sąsiada jest podejmowana na podstawie sprawdzenia przynależności sąsiada do sieci, w której znajduje się dany interfejs.

Natomiast dodawanie interfejsów poprzez specyfikację adresu sieci działa na podobnej zasadzie, czyli po odwzorowaniu wirtualnych nazw interfejsów (Kathara) na fizyczne jesteśmy w stanie ustalić na podstawie adresów sieci, które interfejsy fizyczne będą musiały być dodane do protokołu.

2.4.2 OSPF

Zaimplementowana została również prosta obsługa protokołu OSPF. Podobnie jak w przypadku RIP, nie wszystkie komendy do jego konfiguracji udostępniane są przez Quagga. Według dokumentacji OSPF w protokole netconf [15] , istnieje szereg ustawień, które jednak są dosyć skomplikowane, więc aktualnie ograniczono się jedynie do najbardziej podstawowych ustawień, są to:

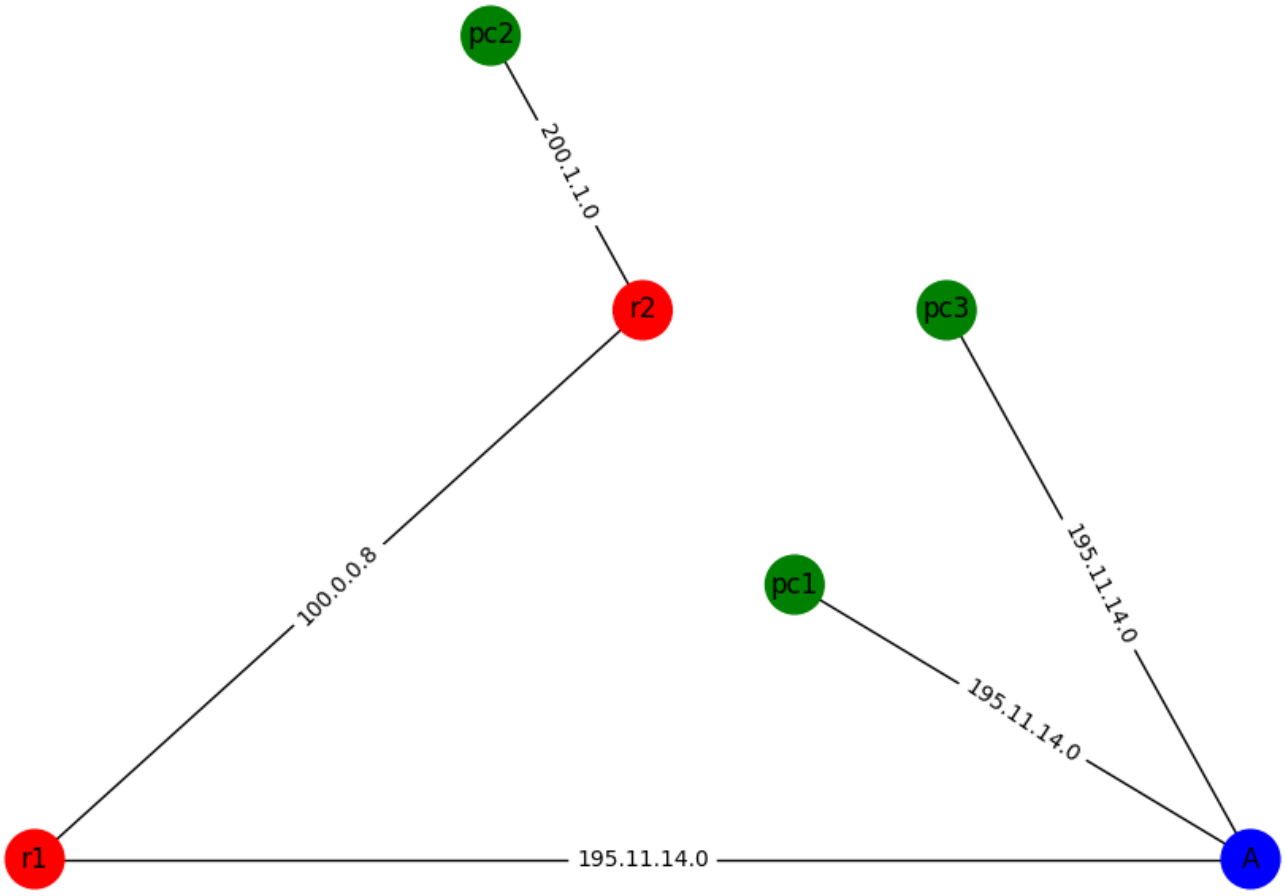
1. Router-id
2. Rozgłaszane sieci:
   (a) Adres IP
   (b) Wildcard mask
   (c) Area

Natomiast w Quagga obsługiwane są następujące komendy:

1. ospf router-id <id> - ustawienie parametru router-id
2. network <ip>/<mask-prefix> area <area> - konfiguracja rozgłaszanej sieci wraz z przypisanym area

2.5 Przykładowa wizualizacja prostej topologii

Program pozwala na wizualizację topologii, zarówno wirtualnej po dokonaniu parsowania danych Kathary, oraz fizyczną po jej odczytaniu z urządzeń sieciowych. Routery zaznaczone są na czerwono, hosty na zielono, a switche na niebiesko.



Do odczytu szczegółów na ten moment sugerujemy używanie debuggera i ustawienie breakpointa w punkcie, w którym topologia została utworzona przez program (docelowo można by było rozwinąć CLI o wypisywanie szczegółów poszczególnych urządzeń).

3 Potencjalnie przydatne informacje

Instalacja Kathary, wykorzystanie CLI oraz tworzenie wirtualnych topologii jest bardzo dobrze opisane w repozytorium Kathara-Labs[4].

Testowaliśmy automatyczne tworzenie topologii w GNS-3, fragment kodu umożliwiający utworzenie topologii typu host - router - router - host dostępny jest w repozytorium projektu[16]. Ponadto GNS-3 umożliwia połączenie się do konsoli urządzenia przez telnet, w naszej implementacji potrzebny był port szeregowy, żeby to osiągnąć można wykorzystać komendę:

```
socat PTY,raw,link=/dev/ttyVUSB0 tcp:127.0.0.1:5001
```

gdzie **/dev/ttyVUSB0** jest utworzonym wirtualnym portem szeregowym, a **127.0.0.1:5001** adresem wirtualnego urządzenia, które chcemy konfigurować.

Jak już wspomnieliśmy, nie mieliśmy opcji przetestowania netconfa w GNS-3. W ograniczonym zakresie (tylko 1 urządzenie i dosyć wolne działanie) można wykorzystać sandbox Cisco devnet[9] (potrzebne konto Cisco), przykładowe testy są dostępne w repozytorium projektu[17].

Do rozbudowy projektu o wsparcie GRE lub firewall mogą się przydać komendy do obsługi sieciowych namespace'ów Linux'a, w szczególności:

```
docker inspect -f '{{.State.Pid}}' <container_id> - uzyskanie PID'u kontenera

ip netns attach <nazwa_nsa> <pid_kontenera> - podpięcie namespace'a kontenera
                                              do hosta by móc z niego korzystać

ip netns exec <nazwa_nsa> <komenda> - wywołanie komendy w podpiętej
            przestrzeni nazw kontenera (komenda to np. ip route, ip link, iptables...)
```

Następnie tunel GRE można utworzyć w sposób opisany w tutorialu[10], należy pamiętać o prefixowaniu komend komendą *ip netns exec*, oczywiście w przypadku Kathary powinna być również opcja bezpośredniego wykorzystania *docker exec -it <container-id> <komenda>*. Powyższe może się przydać jakby zaszła potrzeba tworzenia wirtualnych interfejsów (Linux veth) łączących różne przestrzenie nazw. Do konfiguracji switchy, np. spanning tree, może przydać się narzędzie brctl[12]. Do tworzenia i konfiguracji VXLAN albo VLAN można wykorzystać *ip link*, ogólnie zakres zastosowań tego programu jest ogromny[11].

Źródła i przydatne materiały

[1] *Repozytorium projektu.* https://github.com/Fl0k3n/vtptv.

[2] *Przegląd narzędzi do emulacji sieci.* https://www.brianlinkletter.com/2023/02/network-emulators-and-network-simulators-2023.

[3] *Repozytorium z przykładami Kathary.* https://github.com/KatharaFramework/Kathara-Labs.

[4] *Kathara tutorial.* https://github.com/KatharaFramework/Kathara-Labs/blob/master/001-kathara-introduction.pdf.

[5] *Przykłady wykorzystania Containernet.* https://github.com/containernet/containernet/tree/master/examples.

[6] *Przykłady wykorzystania Minineta do konfiguracji klasycznego routingu.* https://github.com/edwinsc/mininet_ospf_bgp/blob/master/ospf_bgp/start.py.

[7] *Zasady działania minineta.* http://mininet.org/overview/.

[8] *Bindingi GNS-3 do Pythona.* https://github.com/davidban77/gns3fy.

[9] *Router obsługujący netconf w sandboxie Cisco Devnet.* https://devnetsandbox.cisco.com/RM/Diagram/Index/27d9747a-db48-4565-8d44-df318fce37ad?diagramType=Topology.

[10] *Konfiguracja GRE w systemie Linux.* https://www.xmodulo.com/create-gre-tunnel-linux.html.

[11] *Spis możliwości narzędzia ip link.* https://man7.org/linux/man-pages/man8/ip-link.8.html.

[12] *Konfiguracja wirtualnych switchy w systemie Linux.* https://man7.org/linux/man-pages/man8/brctl.8.html.

[13] *Dokumentacja Quagga*. https://www.nongnu.org/quagga/docs/quagga.pdf.

[14] *RFC - RIP netconf*. https://www.rfc-editor.org/rfc/rfc8695.pdf.

[15] *RFC - OSPF netconf*. https://www.rfc-editor.org/rfc/rfc9129.pdf.

[16] *Przykład tworzenia topologii z API GNS3*. https://github.com/Fl0k3n/vtptv/blob/master/playground/gns3/gns3_testing.py.

[17] *Przykład wykorzystania netconf z Cisco devnet*. https://github.com/Fl0k3n/vtptv/blob/master/playground/netconf/testing.py.