Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Instytut Automatyki i Informatyki Stosowanej

Praca inżynierska

Porównanie efektywności protokołów OpenFlow i MPLS

Autor:
Łukasz Fijas

Promotor:
dr inż. Mariusz Kamola

Warszawa 2013

Streszczenie

Celem mojej pracy inżynierskiej było porównanie efektywności działania sieci komputerowych wykorzystujących technologie OpenFlow oraz MPLS. Dodatkowo pokazano w niej przykładowe zastosowanie wspomnianych protokołów w inżynierii ruchu sieciowego. Punktem odniesienia w czasie przeprowadzanych testów były rozwiązania wykorzystujące protokół IP. Na tym tle zostały zaprezentowane możliwości OpenFlow i MPLS. Środowisko testowe zostało stworzone przy użyciu urządzeń dostępnych w wydziałowym laboratorium. W trakcie eksperymentów wykorzystywano programowe routery działające pod kontrolą systemu operacyjnego Linux z dodatkiem MPLS for Linux, routery sprzętowe firmy Cisco oraz przełączniki programowe oparte na Linuksie i zestawie specjalnie zainstalowanych narzędzi.

Abstract

The aim of my bachelor's thesis was to compare efficiency of computer networks, which were using either OpenFlow or MPLS. Moreover, it contains examples how to leverage these technologies to carry out tasks connected with traffic engineering. Capabilities of OpenFlow and MPLS were presented against traffic engineering capabilities provided by IP, which was point of reference during the conducted studies. A set of devices, which were available in laboratory in faculty building, was used to create a test bed. Different types of network appliances were used during the research: software routers running Linux operating system with pre-installed MPLS for Linux patch, Cisco hardware routers and software switches, which were based on Linux with special tools installed.

Spis treści

Rozdział 1

Wstęp

W dzisiejszych czasach ciężko jest już nawet sobie wyobrazić korzystanie z komputera bez połączenia z siecią. Tą najbardziej powszechną i rozwijającą się w ostatnich latach we wprost szalonym tempie, jest sieć Internet. Dodatkowo należy zwrócić uwagę, że obecnie z dostępu do Internetu użytkownicy nie korzystają już tylko i włącznie na komputerach, ale używają do tego telefonów komórkowych, tabletów, aparatów czy nawet lodówek. Bardzo modne jest także pojęcie „chmury", czyli korzystanie z usług i zasobów udostępnianych nam na innych urządzeniach, do których dostęp uzyskujemy właśnie poprzez sieć komputerową. Wszystko to powoduje, że dziedzina informatyki, jaką są sieci komputerowe, jest kluczowa i nieustannie prowadzone są różnego rodzaju badania i eksperymenty, które mają powodować jej rozwój.

W trend tych badań, wpisuje się także moja praca, w ramach której przeprowadziłem serię doświadczeń oraz testów, które miały na celu zbadanie efektywności działania i możliwości zarządzania ruchem sieciowym, jakie zyskujemy stosując protokoły: MPLS oraz OpenFlow. Są to technologie, które w założeniu mają pomóc rozwiązać takie problemy jak: transmisja multimediów w czasie rzeczywistym, dostarczanie usług spełniających kryteria Quality of Service (QoS), budowa skalowalnych wirtualnych sieci prywatnych i ściśle powiązana z poprzednimi zagadnieniami, wydajna i efektywna inżynieria ruchu. Kolejne rozdziały szczegółowo opisują zakres przeprowadzonych przeze mnie prac oraz przedstawiają wyciągnięte wnioski, a także wskazują

na potencjalne zastosowanie otrzymanych wyników.

Rozdział drugi zawiera opis zastosowanych protokołów sieciowych: OpenFlow i MPLS oraz wyjaśnienie, dlaczego warto je stosować jako rozszerzenie technologii IP, a także zwięzły opis środowiska, wykorzystywanego w pracy do badań efektywności ich działania.

Rozdział trzeci przedstawia szczegółowy sposób konfiguracji środowiska tak, aby sprzęt udostępniony w laboratorium mógł być wykorzystywany do przeprowadzania testów wydajności i możliwości protokołów MPLS oraz OpenFlow.

Rozdział czwarty to dokładny opis przeprowadzonych eksperymentów wraz z otrzymanymi rezultatami oraz krótkim komentarzem do nich.

Ostatni, piąty, rozdział jest poświęcony na zaprezentowanie wniosków, które wyciągnąłem na podstawie przeprowadzonych prac.

Do pracy dołączony został także dodatek, gdzie znaleźć można wybrane skrypty lub sekwencje poleceń wykorzystywane do konfiguracji urządzeń.

Rozdział 2

Opis środowiska i wykorzystywanych technologii

Środowiskiem podstawowym, od którego zaczynałem, były komputery PC działające pod kontrolą systemu operacyjnego Linux i protokół komunikacyjny IP. Następnie w kolejnych etapach badań przeprowadzałem różnorakie modyfikacje. Wykorzystywane były routery sprzętowe firmy Cisco. Stosowałem protokół OpenFlow oraz protokół MPLS w dwóch implementacjach: MPLS for Linux i Cisco IOS Multiprotocol Label Switching.

2.1 MPLS i OpenFlow

2.1.1 Braki protokołu IP

Tak jak już wspomniałem we wstępie, sieć Internet ciągle się rozwija, a na przestrzeni ostatnich lat obserwujemy bardzo dynamiczny wzrost jej znaczenia w naszym codziennym życiu. Patrząc na nią od strony technicznej zauważamy, że jest ona oparta na protokole IP. Przy użyciu tej technologii nie jesteśmy jednak w stanie zapewnić świadczenia usług transmisji danych o gwarantowanej jakości [1]. Jest to poważny problem, jeśli chcemy dostarczać poprzez sieć multimedia w wysokiej jakości, spełniając warunki czasu rzeczywistego, a przecież właśnie w ostatnim czasie obserwujemy dynamiczny wzrost zapotrzebowania na tego typu usługi. Wiąże się to z pojawieniem się

telefonii internetowej czy też transmisji telewizyjnych dostępnych w Internecie. W celu rozwiązania tego problemu zaczęto pracować na taką metodą, aby zarówno dane, głos, jak i wideo były transmitowane poprzez sieci telekomunikacyjne w ten sam sposób, w postaci pakietowej. Wymaga to jednak zmodyfikowania architektury sieci, co jest ogólnie określane terminem Next Generation Network.

Kolejnym zagadnieniem, z którym nie do końca możemy sobie poradzić przy użyciu technologii IP, jest stworzenie efektywnych mechanizmów sterowania i zarządzania ruchem, czyli tak zwana inżynieria ruchu. Związane jest to z ograniczeniami stosowanych protokołów routingu, takich jak: OSPF czy RIP, które nie pozwalają na arbitralne definiowanie ścieżek przepływu danych.

Opisywane problemy możemy rozwiązać używając technologii MPLS lub OpenFlow.

2.1.2 MPLS

Protokół MPLS [2] został opracowany pod koniec lat 90-tych XX wieku przez grupę IETF (Internet Engineering Task Force). W założeniu nie ma on zastępować żadnego z już wykorzystywanych protokołów komunikacyjnych, w tym najbardziej powszechnego - Internet Protocol, ale być dla nich uzupełnieniem. MPLS może współpracować z takimi technologiami sieciowymi jak: TCP/IP, ATM, Frame Relay, a także SONET. W swojej pracy skupię się tylko na pokazaniu zastosowania protokołu MPLS w połączeniu z sieciami pracującymi w standardzie TCP/IP.

MPLS jest określany mianem protokołu warstwy 2,5 w modelu ISO OSI (rys. 1). Zgodnie z założeniami powinien on łączyć zalety warstwy łącza danych, czyli wydajność i szybkość oraz warstwy sieciowej, czyli skalowalność.

Dzięki MPLS dostajemy bogatszy zestaw narzędzi do zarządzania siecią i inżynierii ruchu, co umożliwia transmisję pakietów po arbitralnie określonych trasach, które nie są możliwe do zdefiniowania stosując klasyczne protokoły routingu. Ponadto protokół MPLS umożliwia tworzenie wirtualnych

Rysunek 1: MPLS w modelu ISO OSI

sieci prywatnych (VPN).

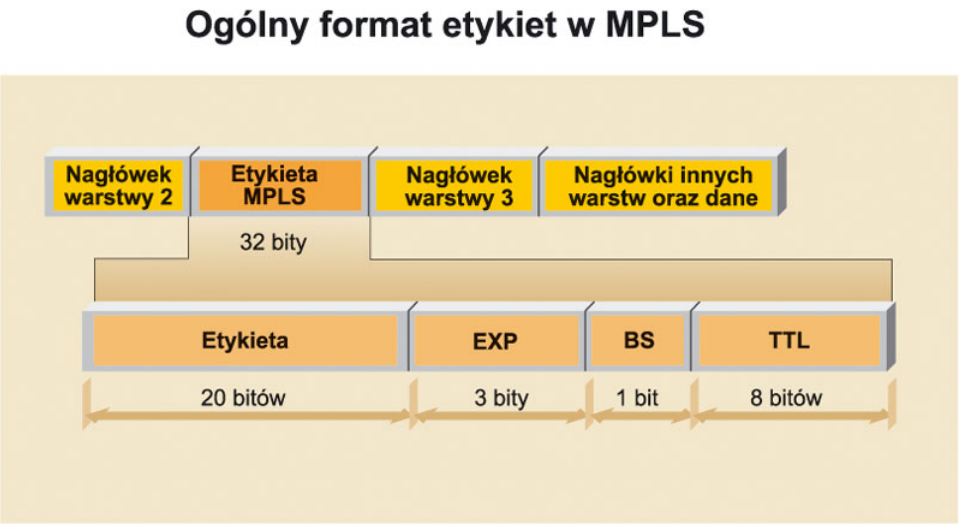
Mechanizm działania MPLS

W domenie MPLS routing IP zastępowany jest mechanizmem przełączania etykiet. Do zrozumienia jego działania niezbędne jest poznanie kilku podanych poniżej pojęć.

LER (Label Edge Router) – router pracujący na brzegu domeny MPLS. Analizuje pakiet IP wchodzący do sieci wykorzystującej MPLS i dodaje do niego nagłówek tego protokołu (ingress router) oraz usuwa nagłówek MPLS z pakietów wychodzących z domeny (egress router).

LSP (Label Switching Path) – ścieżka łącząca dwa routery brzegowe (LER), określająca trasę przepływu pakietów oznaczonych daną etykietą.

LSR (Label Switched Router) – router działający wewnątrz sieci opartej na protokole MPLS. Jest odpowiedzialny za przełączanie etykiet, co pozwala przesyłać pakiet zdefiniowaną ścieżką.



Rysunek 2: Ogólny format etykiet w MPLS, źródło: itpedia.pl

FEC (Forwarding Equivalence Class) – grupa pakietów, które mają takie same wymagania dotyczące ich przenoszenia. Wszystkie pakiety w takiej grupie są traktowane identycznie i podróżują tą samą drogą.

Etykieta MPLS jest dołączana przez LER w chwili wejścia pakietu do sieci wykorzystującej ten protokół. Etykieta o długości 32 bitów jest dodawana pomiędzy nagłówek warstwy drugiej (Ethernet) a nagłówek warstwy trzeciej (IP). Nagłówek MPLS składa się z:

- Pola etykiety etykiety o długości 20 bitów.
- Pola EXP o długości 3 bitów, które określa przynależność do danej klasy ruchu.
- 1-bitowego pola BS, które sygnalizuje osiągniecie dna stosu etykiet (dla najwyższego poziomu pole to przyjmuje wartość 1, dla pozostałych 0).
- Pola TTL (Time To Live), którego funkcja jest analogiczna do pola TTL w nagłówku warstwy 3.

Po dołączeniu etykiety LER przekazuje pakiet do kolejnego węzła zgodnie z odpowiednim wpisem w jego tablicy przełączania etykiet. Każdy router

tworzący LSP ma zdefiniowane odpowiednie reguły zdejmowania etykiet, dołączania nowych oraz ich zamiany i przekazywania pakietów do kolejnych węzłów. Procedura ta jest powtarzana aż pakiet dotrze do ostatniego węzła na trasie (LER), który jest odpowiedzialny za usunięcie etykiety MPLS. Zastąpienie routingu bazującego na analizie nagłówka IP, przełączaniem etykiet o stałej długości powoduje, że podczas określania kolejnych węzłów na trasie pakietu będzie analizowana tylko etykieta MPLS, co teoretycznie powinno przyspieszyć transmisję danych [3]. Technologia MPLS umożliwia budowanie stosu etykiet. W czasie przełączania operacje są zawsze wykonywane na etykietach najwyższego poziomu. Dzięki temu istnieje możliwość definiowania tuneli, co jest przydatne przy tworzeniu wirtualnych sieci prywatnych (VPN).

Quality of Service i inżynieria ruchu

Zestawiając za pomocą MPLS ścieżkę przepływu pakietów jesteśmy w stanie zastosować tzw. ustalone kierowanie. Polega to na tym, że lista węzłów, przez które będzie wędrował pakiet jest ustalana przez LER w momencie wejścia pakietu w obręb domeny MPLS. Dzięki temu jesteśmy w stanie kierować ruch po trasach innych niż, wynikałoby z klasycznego protokołu routingu. W ten sposób możemy wybrać ścieżkę, która posiada zarezerwowane zasoby, spełniające wymogi QoS. Do monitorowania aktualnego stanu łącza możemy posłużyć się protokołami OSPF lub IS-IS i na tej podstawie zestawić adekwatne LSP dla poszczególnych klas ruchu.

Tunelowanie i VPN

Za pomocą MPLS jesteśmy w stanie w bardzo łatwy sposób zestawiać tunele pomiędzy wybranymi węzłami sieci, co można wykorzystać do tworzenia wirtualnych sieci prywatnych (VPN). Wykorzystuje się tutaj możliwości stosu etykiet. Wejściowy LER dodaje do pakietu dwie etykiety MPLS, gdzie pierwsza określa interfejs wyjściowy pracujący w określonej sieci VPN na docelowym routerze brzegowym, a druga definiuje ścieżkę do wyjściowego LER. Dzięki takiemu podejściu jesteśmy wstanie zagwarantować logiczną

separację pomiędzy różnymi sieciami VPN, wykorzystując jedną, wspólną infrastrukturę sieciową. W porównaniu do realizacji sieci VPN opartej na fizycznym wydzieleniu kanałów komunikacji (np. poprzez podział pasma), stosując MPLS zyskujemy dużo większą skalowalność rozwiązania i łatwość dodawania nowych użytkowników. Dodatkowo komunikacja typu każdy użytkownik z każdym jest możliwa bez tworzenia rozległej siatki połączeń typu punkt-punkt.

MPLS – różne implementacje

Charakterystyczną cechą protokołu MPLS jest mnogość jego wersji, które nie zawsze są w pełni zgodne ze standardem, co z kolei powoduje, że mogą nie być kompatybilne. Dostępne są rozwiązania rozwijane na zasadach open source, a także te typowo komercyjne, rozwijane przez takie firmy jak: Cisco, Huawei, Alcatel czy Juniper.

W tej części krótko omówię cztery różne projekty. Z trzema z nich miałem do czynienia podczas badań prowadzonych na potrzeby tej pracy. Ostatecznie, z przyczyn podanych poniżej, wykorzystywane były tylko dwa z wymienionych rozwiązań.

MPLS for Linux MPLS for Linux [4] jest to projekt open source, który w swoim założeniu ma dodać do jądra systemu operacyjnego Linux obsługę protokołu MPLS. Twórcy projektu udostępniają uaktualnienie do jądra Linuxa, które umożliwia wykorzystanie protokołu MPLS. W ten sposób zwykły komputer PC może być używany jako router posługujący się MPLS. W obecnej chwili projekt nie jest intensywnie rozwijany, co objawia się między innymi tym, że jest on zgodny z wersją jądra systemu Linux oznaczoną jako 2.6.x. Jego zaletą jest jednak stabilność działania i dlatego był wykorzystywany w czasie testów.

MPLS Implementation for the Linux Kernel Projekt [5], który jest rozwijany pod kierownictwem Igora Maravica, ma na celu dostarczenie jądra systemu Linux obsługującego komunikację przy użyciu protokołu MPLS[1].

[1] github.com/i-maravic/MPLS-Linux

Projekt jest niejako kontynuacją prac prowadzonych w ramach implementacji MPLS for Linux, gdyż bazuje na przedstawionych tam rozwiązaniach. Wydawać by się mogło, iż to, że jest on na bieżąco rozwijany i wykorzystuje jądro Linuxa w wersji 3.x powinno być jego atutem i dlatego też próbowałem używać go w moich pracach. Szybko jednak wycofałem się z tego pomysłu z powodu bardzo uciążliwej niestabilności zbudowanych systemów, które często odmawiały współpracy.

IOS Multiprotocol Label Switching Cisco IOS Multiprotocol Label Switching [6] jest to implementacja protokołu MPLS stworzona przez firmę Cisco. Jest to technologia komercyjna, w przeciwieństwie do wolnego oprogramowania jakim są rozwiązania przedstawione powyżej. Producent reklamuje ją jako doskonałe rozwiązanie, na którym można oprzeć budowę inteligentnych sieci następnej generacji. Zgodnie z wymaganiami wersja protokołu stworzona przez Cisco może pracować, jako rozszerzenie do IP, ATM, Frame Relay i Ethernet. Technologia ta jest obsługiwana przez szeroką gamę urządzeń wytwórcy.

DRAGON Dynamic Resource Allocation via GMPLS Optical Networks (DRAGON) jest projektem naukowo-badawczym rozwijanym przez grupę współpracujących, amerykańskich uniwersytetów i instytucji. Jego celem jest efektywne wykorzystanie możliwości, jakie dają sieci optyczne i dostarczenie oprogramowania pozwalającego na korzystanie z różnorodnej infrastruktury sieciowej w taki sposób, aby wydajnie transmitować dane w sieciach IP. W ramach tego projektu opracowano i stworzono eksperymentalną sieć komputerową, która pracuje w obrębie Waszyngtonu i okolic. Oparta jest ona na rozwiązaniu GMPLS (Generalized Multiprotocol Label Switching) i wykorzystuje zarówno elementy sieci pakietowych, jak i łączy komutowanych.

2.1.3 OpenFlow

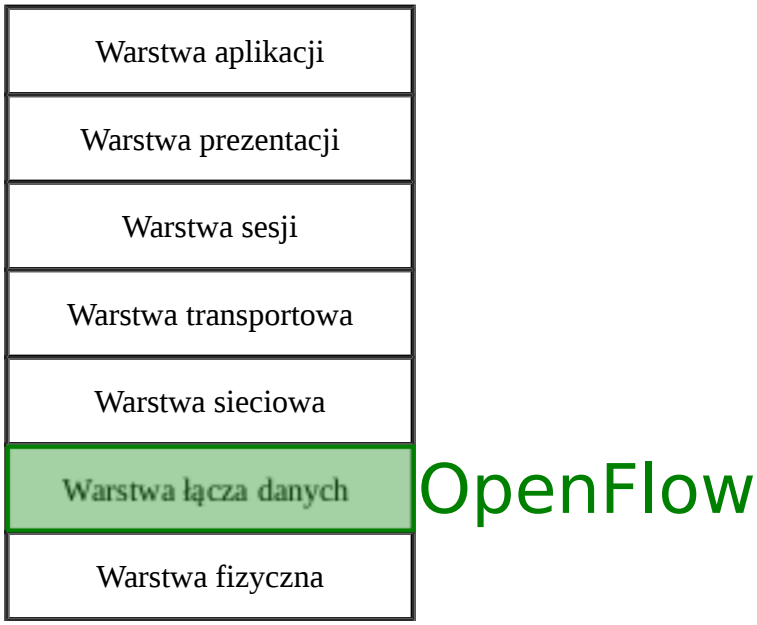
OpenFlow [7] jest to projekt open source, który zrodził się w połowie pierwszej dekady XXI wieku na amerykańskich uniwersytetach Stanford i Ka-

lifornia. Jego pierwsza oficjalna specyfikacja została ogłoszona pod koniec 2009 roku i została oznaczona jako wersja 1.0.0. Obecnie opiekę na jego rozwojem sprawuje organizacja Open Networking Foundation, a ostatnią wersją, ogłoszoną w kwietniu tego roku, jest 1.3.2. Powstanie protokołu OpenFlow jest ściśle związane z pojęciem Software-defined Networking, czyli programowym definiowaniem pracy sieci komputerowych. Pojęcie to również zostało wymyślone na Stanford University, mniej więcej w tym samym okresie, co sam protokół. W skrócie koncepcja ta zakłada odseparowanie wysokopoziomowej kontroli nad przepływem ruchu w sieci (ang. control plane) od sprzętowych rozwiązań przekazywania pakietów (ang. data plane). Dzięki takiemu podejściu jesteśmy w stanie zarządzać ruchem w sieciach komputerowych w ten sam sposób, niezależnie od używanego sprzętu.

Wartość dodana, jaką uzyskujemy stosując OpenFlow, jest podobna do tej oferowanej przez MPLS. Wykorzystując technologię OpenFlow, otrzymujemy bogaty zestaw narzędzi pozwalający na zaawansowaną inżynierię ruchu. Możemy optymalizować transmisję pod kątem zapewnienia odpowiedniego pasma, unikania opóźnień czy liczby węzłów, przez które przechodzi pakiet. Dzięki temu jesteśmy w stanie oferować usługi spełniające wymagania QoS.

W oparciu o OpenFlow jesteśmy w stanie tworzyć wirtualne sieci prywatne, a nawet coś więcej, bo sieci wielodzierżawne [8]. Protokół ten wprowadza pojęcie przepływów ruchu (ang. flows), co pozwala na całkowitą wirtualizację sieci i separację ruchu. Daje to administratorom nowe możliwości rozbudowy, modyfikacji i testowania sieci komputerowych. Mogą oni badać nowe rozwiązania wykorzystując do tego środowisko produkcyjne, a jednocześnie nie zakłócają w ten sposób normalnego działania sieci. Sprawia to, że idealnym miejscem do zastosowania protokołu OpenFlow są między innymi sieci akademickie. Ta cecha tej technologii jest ściśle związana z historią rozwoju protokołu, która swój początek miała na słynnych amerykańskich uniwersytetach.

OpenFlow nie jest rewolucyjnym rozwiązaniem, jeśli weźmiemy pod uwagę możliwości jego poprzednika, czyli MPLS. Powstaje więc pytanie w jakim celu nowa technologia jest tak intensywnie rozwijana? Jej twórcy przekonują, że dzięki jednemu ściśle określonemu standardowi jesteśmy w stanie progra-

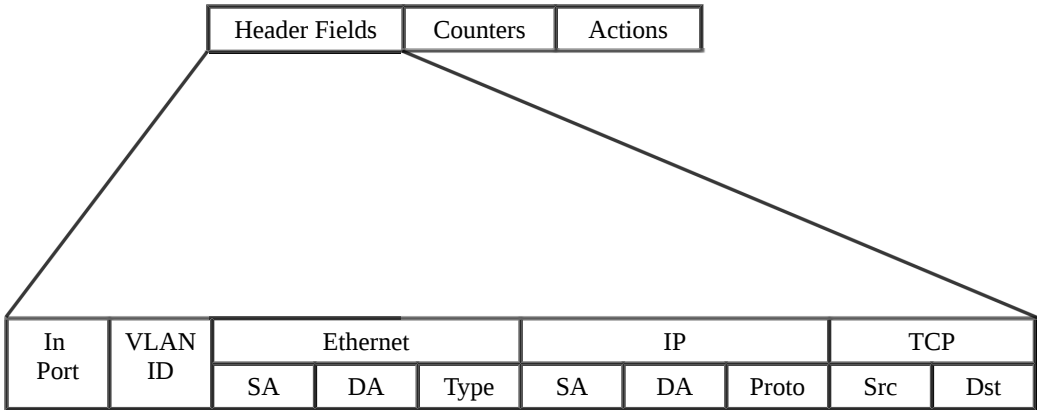


Rysunek 3: OpenFlow w modelu ISO OSI

mowo definiować działanie sieci, całkowicie niezależnie od tego, jakiego producenta sprzęt został użyty. Protokół MPLS pomimo tego, że też ma określoną specyfikację, to jednak często jest odmiennie implementowany przez różnych dostawców. Dodatkowo, zgodnie z zapowiedziami, protokół OpenFlow jest prostszy od swojego poprzednika. Niewątpliwie zaletą OpenFlow jest także to, że w chwili obecnej jego rozwojem zajmuje się Open Networking Foundation, czyli organizacja powołana do życia dzięki porozumieniu najbardziej liczących się graczy w dziedzinie technologii komputerowych, takich jak: Facebook, Google, Yahoo, Microsoft, Dell, IBM, HP, Cisco i wielu innych. Jest to rozwiązanie open source wspierane przez coraz szerszą gamę dedykowanych urządzeń sieciowych, ale także może ono działać na zwykłych komputerach PC z zainstalowanym systemem operacyjnym Linux.

Mechanizm działania OpenFlow

OpenFlow jest protokołem działającym w drugiej warstwie modelu ISO OSI (rys. 3), co odróżnia go od protokołu MPLS, który działa zarówno w warstwie łącza danych, jak i sieciowej. Urządzenia sieciowe, które działają w war-



Rysunek 4: Budowa tabeli przepływów

stwie łącza danych, to dzisiaj najczęściej przełączniki. Do stworzenia sieci opartej na technologii OpenFlow niezbędne są trzy elementy: przełączniki współpracujące z tym protokołem, kontroler oraz kanał komunikacyjny, wykorzystujący protokół OpenFlow, za pomocą którego kontroler i przełącznik mogą się komunikować.

Przełącznik OpenFlow Jest to element infrastruktury sieciowej działający w warstwie drugiej modelu ISO OSI, który przechowuje w swojej pamięci tabelę przepływów (ang. flow table) oraz posiada kanał komunikacyjny, który może być używany do komunikacji za pomocą protokołu OpenFlow z kontrolerem. Możemy wyróżnić specjalistyczne, fizyczne urządzenia, czyli dedykowane przełączniki obsługujące tę technologię, jak i przełączniki programowe, czyli zazwyczaj zwykłe komputery, tyle że działające pod kontrolą odpowiednio przygotowanego systemu operacyjnego. Na dokładniejsze omówienie zasługuje pojęcie tabeli przepływów (rys. 4). Składa się ona z trzech kolumn:

1. pola identyfikującego wpis (ang. match fields), które jest tworzone na postawie nagłówka pakietu,

2. licznika (ang. counters), gdzie znajdują się informacje statystyczne takie jak: liczba przesłanych pakietów i bajtów oraz czas od pojawienia się ostatniego pakietu pasującego do danej reguły,

3. tzw. pola akcji (ang. actions), które określa sposób przetwarzania pakietu.

Wpisy do niej są dodawane za pośrednictwem kontrolera. Określają one, jak ma się zachowywać przełącznik po otrzymaniu pakietu spełniającego dany warunek dopasowania. Przełącznik może przychodzące dane przesłać na dany port wyjściowy, odrzucić je lub przesłać do kontrolera. Ta ostatnia akcja jest najczęściej wykonywana, gdy przełącznik po otrzymaniu pakietu nie jest w stanie dopasować go do żadnej z istniejących reguł. W takiej sytuacji to kontroler podejmuje odpowiednią decyzję, jaką regułę dodać do tabeli przepływów tak, aby akcja w stosunku do następnych takich samych pakietów była podejmowana już tylko przez przełącznik.

Kontroler Za jego pomocą do tabel przepływów w przełącznikach dodawane są odpowiednie reguły. Funkcję kontrolera może pełnić jakiś bardzo wyrafinowany program, sterujący ruchem wedle określonych kryteriów bądź bardzo prosty mechanizm, który będzie dodawał statyczne wpisy do pamięci przełącznika.

Kanał komunikacyjny Jest to bardzo ważny element w sieciach opartych na technologii OpenFlow. Służy on do komunikacji przełączników z kontrolerami, co jest kluczowe, ponieważ tak naprawdę decyzje o zarządzaniu ruchem w sieci są podejmowane przez kontroler i muszą być rozpropagowane wśród przełączników. Postać danych przesyłanych takim kanałem musi być zgodna ze specyfikacją OpenFlow i zazwyczaj jest zaszyfrowana przy użyciu SSL. Specyfikacja OpenFlow wyróżnia trzy rodzaje komunikacji [7]. Są to:

1. Komunikacja typu „kontroler do przełącznika” (ang. controller-to-switch), która polega na tym, że kontroler wysyła wiadomość do przełącznika i ewentualnie otrzymuje odpowiedź. Protokół przewiduje sześć rodzajów takich wiadomości: utworzenie sesji (ang. features), konfiguracja (ang. configuration), modyfikacja stanu (ang. modify-state), odczyt stanu (ang. read-state), wysłanie pakietu (ang. send-packet) oraz potwierdzenie (ang. barrier).

2. Łączność asynchroniczna jest inicjowana przez przełączniki i wyróżniamy tu cztery podstawowe rodzaje wiadomości: otrzymanie pakietu (ang. packet-in), usunięcie wpisu z tablicy przepływów (ang. flow-removed), status portu (ang. port-status), komunikat błędu (ang. error)

3. Komunikacja symetryczna (ang. symmetric) może być inicjowana przez dowolną ze stron, a wiadomości przesyłane w ten sposób to: przywitanie (ang. hello), echo (ang. echo) oraz „vendor" (ang. vendor) – typ wiadomości zarezerwowany do użycia w przyszłości.

2.2 Środowisko w laboratorium

Wszelkie testy możliwości i efektywności komunikacji z wykorzystaniem wybranych protokołów sieciowych były przeprowadzane przy użyciu fizycznego sprzętu dostępnego w laboratorium na wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. W eksperymentach wykorzystywano routery Cisco 2801, komputery PC oraz laptop Dell Inspiron N5010.

2.2.1 Routery Cisco 2801

Router Cisco 2801 jest to urządzenie serii 2800, którą tworzą cztery modele. Sprzęt ten cechuje duża elastyczność w konfiguracji, która pozwala na dostarczanie za jego pomocą szerokiego wachlarza usług, co powoduje, że są powszechnie stosowane w rozwiązaniach biznesowych. Sprzęt, z którego korzystam, ma dwa porty LAN pracujące w standardzie FastEthernet oraz szereg gniazd kart interfejsów, pozwalających na dołączenie modułów spełniających indywidualne wymagania użytkowników. Model 2801 może być zamontowany w szafie rackowej, gdzie będzie zajmował 1 jednostkę, ponieważ jego wysokość to 1,75 cala. Schematyczny wygląd takiego urządzenia jest zaprezentowany na rysunku (rys. 5).

Wykorzystanie w mojej pracy akurat tego typu routera, było podyktowane tym, że kilka takich urządzeń zostało mi udostępnionych w wydziałowym laboratorium.

Rysunek 5: Router Cisco 2801, źródło: www.cablesandkits.com

Używane przez mnie routery pracowały pod kontrolą systemu operacyjnego Internetwork Operating System w wersji 12.4T Advanced Enterprise Services, przeznaczonej dla modelu 2801. Nomenklatura wersji systemu IOS jest dość zawiła i skomplikowana, a jest to spowodowane dużą liczbą gałęzi i paczek, które tworzą ten system operacyjny. Symbol „T" w nazwie oznacza, że jest to gałąź „Technology" systemu, która w czasie rozwoju danej wersji zawiera nowo opracowane funkcje i dlatego może nie być stabilna. Natomiast „Advanced Enterprise Services" jest to nazwa paczki i określa ona, jakie funkcje są dostępne w danym obrazie systemu IOS. „Advanced Enterprise Services" jest to pełna wersja systemu operacyjnego zawierająca wszystkie dostępne funkcje, dzięki czemu mogłem uzyskać dokładnie taką konfigurację środowiska, jakiej potrzebowałem.

2.2.2 Komputery PC i laptop

Dostępne w wydziałowym laboratorium komputery, które były wykorzystywane w pracy, to urządzenia posiadające dwurdzeniowy procesor AMD

Athlon 64 X2 Dual Core Processor 4000+ taktowany zegarem o częstotliwości 2,1 GHz, 2 GB pamięci RAM oraz w zależności od egzemplarza jedną, dwie lub trzy karty sieciowe.

Do przeprowadzania testów używałem także prywatnego laptopa Dell Inspiorn N5010 z dwurdzeniowym procesorem Intel Core i3 2.40 GHz, 4 GB pamięci RAM oraz jedną, wykorzystywaną kartą sieciową (karta typu WLAN nie była używana).

2.2.3 Przełączniki 3Com Switch 4200 26-Port

Z powodu poważnego ograniczenia, jakim był fakt, że każdy z dostępnych w laboratorium routerów Cisco dysponuje tylko dwoma portami FastEthernet, musiałem w trakcie przeprowadzania niektórych eksperymentów używać dodatkowych przełączników. Wykorzystałem do tego celu dwudziestosześcioportowe przełączniki firmy 3Com. Jest to, jak zapewnia producent, wysokiej klasy sprzęt, który łączy wydajność i prostotę obsługi.

2.3 Generatory ruchu

Głównym tematem pracy jest sprawdzenie efektywności protokołów OpenFlow i MPLS. Aby zbadać jak zmienia się wydajność pracy sieci w zależności od zastosowanego rozwiązania, należało wygenerować w niej „sztuczny" ruch. W tym celu użyłem programowych generatorów ruchu: *mgen* i *iperf*, które są udostępniane jako wolne oprogramowanie.

2.3.1 Mgen

Multi-generator, czyli w skrócie *mgen*[2] jest to programowy generator ruchu stworzony i rozwijany przez amerykańską instytucję o nazwie Naval Research Laboratory. Służy do przeprowadzania testów wydajności sieci komputerowych. Narzędzie to potrafi wytwarzać ruch pakietów TCP i UDP oraz zapisywać odpowiednie statystyki. *Mgen* dostarcza bardzo szeroki zestaw

[2] cs.itd.nrl.navy.mil/work/mgen/index.php

możliwości, jeśli chodzi o rodzaj generowanego ruchu, dzięki czemu jesteśmy w stanie zaplanować scenariusze testowe odpowiadające rzeczywistym warunkom panującym w środowisku roboczym. Wykorzystywana przez mnie wersja to 5.02, która może być uruchamiana na systemach Linux, Windows oraz MacOS X. Program można pobrać w postaci archiwum ze strony internetowej Naval Research Laboratory, a także jest on dostępny jako gotowa paczka w repozytoriach systemu Ubuntu.

2.3.2 Iperf

Iperf[3] jest narzędziem do testowania sieci komputerowych stworzonym przez grupę Distributed Applications Support Team działającą w amerykańskim National Laboratory for Applied Network Research. Program został napisany w języku C++ i pracuje w architekturze klient-serwer. Do badania wydajności wykorzystuje wygenerowane strumienie pakietów TCP i UDP, a parametry sieci, które przy jego pomocy możemy obserwować, to przepustowość łącza, zmienność opóźnienia (jitter) oraz liczba utraconych pakietów. Używana przeze mnie wersja programu oznaczona została jako 2.0.5 i jest dostępna w repozytoriach Ubuntu jako paczka do pobrania i instalacji. Istnieje także wersja działająca w systemach z rodziny Microsoft Windows.

[3] iperf.sourceforge.net

Rozdział 3

Przygotowanie i konfiguracja środowiska

W momencie określania tematyki i zakresu tej pracy, została podjęta decyzja, że wszelkie próby badania efektywności i możliwości opisywanych protokołów sieciowych będą miały miejsce w rzeczywistym środowisku w wydziałowym laboratorium sieci komputerowych. Rozwiązaniem alternatywnym było stworzenie środowiska w oparciu o maszyny wirtualne, które mogłyby być uruchamiane właściwie na dowolnym komputerze. Za tym drugim rozwiązaniem przemawia niewątpliwie wygoda i brak kłopotów z dostępem do fizycznego sprzętu. Zdecydowałem się jednak na pracę na sprzęcie dostępnym na uczelni z kilku powodów. Po pierwsze konfiguracja i testy środowiska, w którym wykorzystywany jest fizyczny sprzęt, jest bliższa temu co spotykamy obecnie w rzeczywistych zastosowaniach sieciowych, gdzie wirtualizacja jest raczej rzadkością. Stosowana w pracy konfiguracja może odgrywać rolę wycinka większej sieci, co powoduje, że otrzymane wyniki mogą łatwiej znaleźć zastosowanie w praktyce. Po drugie, badając efektywność protokołów, niewątpliwie bardziej wiarygodne wyniki można otrzymać wykorzystując realną infrastrukturę niż emulując działanie urządzeń za pomocą maszyn wirtualnych. Po trzecie w laboratorium były dostępne dedykowane urządzenia sieciowe firmy Cisco, które mogą obsługiwać protokół MPLS. Produkty tej firmy są powszechnie wykorzystywane w rozwiązaniach komercyjnych, dla-

tego warto jest sprawdzić, co możemy zyskać stosując je do budowy infrastruktury sieciowej.

W celu wykorzystania komputerów w roli programowych przełączników czy routerów, przygotowałem odpowiednie wersje systemu Linux i umieściłem je na pamięciach USB. Wykorzystywane komputery mają możliwość uruchamiania się z pendrive'ów. W ten sposób, nie ingerując w oprogramowanie, które jest potrzebne innym do codziennej pracy w laboratorium, mogłem korzystać z systemów operacyjnych, które spełniają niezbędne wymagania.

Kolejne podrozdziały zawierają szczegółowy opis przygotowania, konfiguracji i instalacji systemów operacyjnych oraz niezbędnego oprogramowania, które było wykorzystywane w czasie testów.

3.1 MPLS for Linux – routery programowe

Przygotowania mające na celu umożliwienie używania komputerów PC jako routerów programowych polegały na skompilowaniu, odpowiednio wcześniej skonfigurowanego, jądra systemu Linux oraz instalacji niezbędnych narzędzi. Początkowo, jak już o tym wspomniałem w części 2.1.2, próbowałem korzystać z wersji jądra przygotowanej w ramach projektu prowadzonego przez Igora Maravica [5]. Nie udało mi się jednak w ten sposób osiągnąć ustawień, przy których sieć działałby w sposób, jaki był konieczny do przeprowadzenia zaplanowanych eksperymentów. Problemem okazało się częste niepoprawne działanie systemu oraz jego duża niestabilność, która ujawniała się w momencie prób konfiguracji przygotowującej komputery do pełnienia roli routerów programowych.

Wobec zaistniałej sytuacji w swojej pracy postanowiłem wykorzystywać oprogramowanie udostępniane przez twórców starszej, nierozwijanej już wersji protokołu MPLS przeznaczonej dla systemu Linux. W tym przypadku kłopotliwy okazał się z kolei wybór odpowiedniej wersji jądra systemu operacyjnego i zgodnego z nim zestawu narzędzi obsługujących MPLS. Bardzo pomocna w tej sytuacji okazała się praca dyplomowa napisana przez Michała Kisiela [9], w której również korzystano z technologii MPLS for Linux. Znalezione tam informacje sprawiły, że ostatecznie w ramach mojej pracy używane

było jądro systemu Linux oznaczone numerem 2.6.27.24 z zainstalowanym dodatkiem, umożliwiającym obsługę protokołu MPLS, oznaczonym numerem 1.963. W dalszej części szczegółowo opisano kolejne kroki prowadzące do otrzymania oprogramowania, które było wykorzystywane przez routery programowe w czasie przeprowadzanych testów.

Przygotowania systemu należy rozpocząć od instalacji standardowej wersji systemu Debian, a następnie pobrać jądro w wersji 2.6.27.24 oraz dodatek MPLS w wersji 1.963. Proces konfiguracji poprzedzającej kompilację, na którym bazowałem, jest dość dokładnie opisany w dokumentacji dostępnej na stronie domowej projektu [4], dlatego nie będzie tutaj szerzej omawiany. Do kompilacji wykorzystywano zestaw narzędzi dostępnych w systemie Debian o nazwie *kernel-package*. Proces budowania należy rozpocząć, będąc w katalogu zawierającym źródła jądra, za pomocą komendy:
 # make-kpkg –initrd –revision=786:MPLS2.6.27.24 kernel_image kernel_headers.
W wyniku udanej kompilacji powstają dwa nowe pakiety instalacyjne w formacie deb, które należy zainstalować:
 # dpkg -i linux-image-2.6.27.24_MPLS2.6.27.24_i386.deb
 # dpkg -i linux-headers-2.6.27.24_MPLS2.6.27.24_i386.deb.
Na tym etapie dysponujemy już działającym systemem operacyjnym, który ma włączoną obsługę protokołu MPLS wewnątrz swojego jadra. Kolejnym krokiem jest instalacja wersji programu *iproute2* dostosowanej do technologii MPLS. Jest ona dostępna na stronie projektu MPLS for Linux.
 # dpkg -i iproute-2.6.39-mpls_i386.deb

Tak przygotowany komputer może działać jako router programowy w sieci opartej na protokole MPLS. Aby móc wykorzystywać przygotowany system na komputerach znajdujących się w laboratorium należy skopiować go na pamięci USB[1].
 # rsync -aAXv / /mnt –exclude={/dev/*, /proc/*, /sys/*, /tmp/*, /run/*, /mnt/*, /media/*, /lost+found}*

[1]Zakładamy, że pamięć USB jest zamontowana w katalogu /mnt

3.2 Cisco IOS MPLS – routery sprzętowe

Routery Cisco 2801, dostępne w laboratorium, to urządzenia udostępniające funkcję obsługi protokołu MPLS. Niezbędne do tego jest jednak odpowiednie oprogramowanie. W pracy zaplanowano użycie funkcji *mpls static binding* oraz *mpls static crossconnect*. Aby móc z nich korzystać zainstalowałem wersję systemu IOS 12.4T Advanced Enterprise Services, przeznaczoną dla modelu 2801. Przedstawię teraz kolejne kroki tej procedury. Są one oparte na dokumencie udostępnionym przez firmę Cisco [10].

Pierwszym krokiem w procesie instalacji nowej wersji systemu jest uruchomienie serwera TFTP na komputerze, na którym znajduje się odpowiedni obraz systemu. Następnie należy zapewnić komunikację pomiędzy serwerem i routerem, na przykład za pomocą połączenia sieciowego z wykorzystaniem, któregoś z interfejsów FastEthernet. Nowy obraz systemu operacyjnego kopiujemy na kartę pamięci flash. System Cisco IOS wyróżnia dwa tryby pracy: zwykłego użytkownika i użytkownika uprzywilejowanego. Wszystkie polecenia konfiguracyjne i administracyjne musimy wykonywać, będąc w trybie uprzywilejowanym. Polecenie, które pozwoli nam na stanie się użytkownikiem uprzywilejowanym to:
Router>enable.
Teraz, aby zapewnić odpowiednią ilość miejsca w pamięci flash, usuwamy jej dotychczasową zawartość.
Router#erase flash:
Właściwa operacja kopiowania odbywa się za pomocą:
Router#copy tftp: flash:.
Należy podać adres lub nazwę serwera, na którym znajduje się obraz sytemu oraz o nazwę pliku źródłowego i docelowego. Po zakończonym procesie kopiowania, możemy zweryfikować poprawność nowego pliku poleceniem:
Router#verify flash:nazwa_pliku.
Aby router zaczął korzystać z nowej wersji systemu IOS, należy jeszcze zapisać nową konfigurację i przeładować router.
Router#write memory
Router#reload

Taka konfiguracja routera powoduje, że ustawiając opcje obsługi protokołu MPLS mamy dostęp do funkcji *mpls static binding* oraz *mpls static corssconnect*.

3.3 OpenFlow – przełączniki programowe

W celu wykorzystania komputerów PC jako przełączniki programowe, użyłem systemu operacyjnego Linux, a dokładniej jego dystrybucji Ubuntu w wersji 12.04 LTS „Przyjazny puchacz". System wykorzystywał jądro o sygnaturze 3.2.0-31. Szczegółowa instrukcja instalacji narzędzi, które pozwalają na obsługę protokołu OpenFlow znajduje się na stronie domowej projektu, a konkretnie jest opisana w dokumencie [11].

Procedura instalacji wszystkiego, co niezbędne, aby zacząć używać komputera w roli przełącznika wykorzystującego technologię OpenFlow jest następująca. Zacząć należy od pobrania źródeł projektu, które są dostępne w postaci skompresowanego archiwum tar.

$ wget http://openflow.org/downloads/openflow-1.0.0.tar.gz

Po rozpakowaniu przechodzimy do głównego katalogu projektu i kompilujemy jego zawartość przy użyciu programu make.

$ tar xzf openflow-1.0.0.tar.gz
$ cd openflow-1.0.0
$ ./configure
$ make
$ sudo make install

Jeżeli podczas całego procesu nie wystąpiły żadne błędy i wszystko przebiegło pomyślnie, to powinniśmy mieć już gotowy system operacyjny, który umożliwia wykorzystywanie komputera jako przełącznik obsługujący protokół OpenFlow.

System operacyjny przygotowany według powyższej instrukcji skopiowano do pamięci USB za pomocą programu *rsync*, wydając w trybie root polecenie[2]:

[2]Zakładamy, że pamięć USB jest zamontowana w katalogu /mnt

 # rsync -aAXv / /mnt –exclude={/dev/*, /proc/*, /sys/*, /tmp/*, /run/*, /mnt/*, /media/*, /lost+found}*

Mając tak przygotowaną wersję Ubuntu, zainstalowaną na pendrive'ach, można zacząć używać komputerów w laboratorium w roli przełączników programowych.

Rozdział 4

Testy efektywności i możliwości protokołów MPLS i OpenFlow

W tym rozdziale szczegółowo opisano przeprowadzone testy, które mają pokazać zalety bądź wady wyżej wspomnianych rozwiązań. Zawarte są w nim także otrzymane w wyniku eksperymentów rezultaty, które w dalszej części posłużą do sformułowania adekwatnych wniosków na temat użyteczności omawianych technologii. Tezą, którą przeprowadzone próby mają zweryfikować, jest stwierdzenie, że używając czy to OpenFlow, czy MPLS jesteśmy w stanie wydajniej zarządzać ruchem w sieci, nie zmniejszając przy tym jakości dostarczanych usług, a nawet ją poprawiając.

Zrealizowane eksperymenty można zasadniczo podzielić na cztery główne grupy:

1. Sprawdzenie kompatybilności protokołu MPLS w implementacji MPLS for Linux z wersją Cisco IOS Multiprotocol Label Switching.

2. Porównanie efektywności działania sieci komputerowej opartej na standardowym routingu IP z rozwiązaniami: MPLS for Linux, Cisco IOS Multiprotocol Label Switching oraz OpenFlow.

3. Zastosowanie nietrywialnej inżynierii ruchu z użyciem opisywanych technologii.

4. Pokazanie, jakie są możliwości reakcji na awarię pewnych części sieci w przypadku stosowania protokołu MPLS.

Kolejne części tego rozdziału zawierają dokładny opis zagadnień wspomnianych w punktach powyżej.

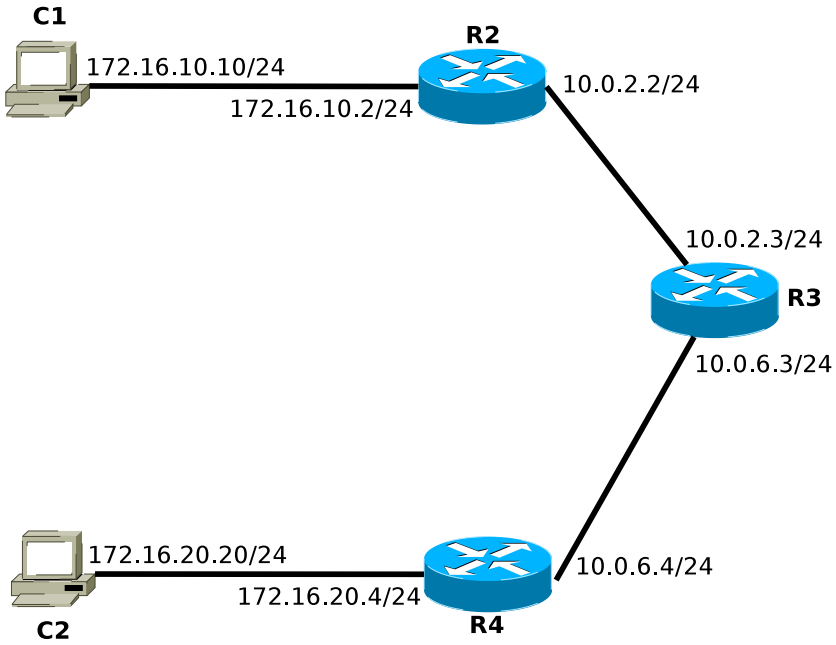
4.1 Kompatybilność różnych implementacji protokołu MPLS

Dysponując przygotowanymi wcześniej urządzeniami, z których jedne to routery programowe używające rozwiązania MPLS for Linux, a drugie to routery sprzętowe Cisco z systemem operacyjnym obsługującym technologię MPLS, należało sprawdzić czy istnieje między nimi możliwość współpracy. W tym celu stworzyłem prostą sieć komputerową składającą się z pięciu węzłów, z których trzy pełniły rolę routerów, a dwa były standardowymi komputerami niewymagającymi żadnego specjalnego oprogramowania. Schematycznie zostało to przedstawione na rysunku 6.

Na wspomnianej grafice widać, że urządzenia pracują w czterech różnych podsieciach. Celem jest zapewnienie komunikacji przy użyciu MPLS pomiędzy dwoma węzłami brzegowymi, którymi są zwykłe komputery PC. Pomijając niewielką liczbę zestawionych przeze mnie węzłów, taka konfiguracja jest często spotykana w rzeczywistych rozwiązaniach komercyjnych, gdzie sieć operatorska wykorzystuje technologię przełączania etykiet MPLS, a z punktu widzenia użytkownika końcowego jest to zupełnie niewidoczne. Tak więc taka koncepcja może obrazować pewien fragment sieci, który jest stosowany przez dostawców usług internetowych (ang. Internet Service Provider).

4.1.1 Konfiguracja

Na schemacie 6 routery zostały oznaczone odpowiednio symbolami R2, R3, R4, a komputery C1 i C2. Aby prawidłowo ustawić działanie komputerów



Rysunek 6: Kompatybilność różnych implementacji protokołu MPLS – schemat sieci

do tej próby, należy na każdym z nich wykonać dwa polecenia[1].

- C1

  # ifconfig eth0 172.16.10.10 netmask 255.255.255.0 up
  # ip route add 172.16.20.0/24 via 172.16.10.2

- C2

  # ifconfig eth0 172.16.20.20 netmask 255.255.255.0 up
  # ip route add 172.16.10.0/24 via 172.16.20.4

Polecenia te ustawiają właściwe adresy IP na interfejsach sieciowych komputerów oraz dodają statyczne wpisy do tablicy routingu. Reguły te zapewniają, że ruch do wybranych adresatów jest kierowany na urządzenia pełniące rolę routerów. Na routerach brzegowych, opisanych jako R2 i R4, ma miejsce dołączanie i zdejmowanie etykiet protokołu MPLS. Router R3 pełni funkcję

[1]Wszystkie polecenia należy wykonywać, będąc zalogowanym jako *root*. Nazwy interfejsów mogą się różnić w zależności od używanego obrazu systemu operacyjnego.

zamieniającego etykiety. Po otrzymaniu pakietu szuka odpowiedniej reguły w tabeli przełączania etykiet i jeżeli właściwy wpis zostanie odnaleziony, to wykonuje akcję do niego przypisaną, czyli w tym przypadku zamienia etykiety MPLS i przekazuje pakiet do kolejnego węzła.

W czasie tego testu używałem jednego routera sprzętowego firmy Cisco oraz dwóch routerów programowych wykorzystujących MPLS for Linux. Działanie sieci było sprawdzane zarówno wtedy, gdy router Cisco pracował jako router brzegowy, jak i wtedy, gdy był węzłem środkowym. Konfiguracja routerów programowych została oparta na dokumentacji dostępnej na stronie projektu[2]. Przykładowe skrypty dla routerów programowych oraz sekwencje poleceń konfigurujące urządzenia Cisco za pomocą, których można odtworzyć dowolny z używanych w tym teście wariantów, są dostępne w dodatku A (Listingi: A.1, A.2, A.3, A.4).

4.1.2 Wyniki

Założeniem tego testu było zbadanie czy tworząc sieć opartą na protokole MPLS, można stosować jednocześnie urządzenia wykorzystujące komercyjną oraz otwartą implementację tej technologii. Stworzona przez mnie konfiguracja dała jednoznaczną odpowiedź, że można. Niezależnie od tego, które urządzenia będą pełniły rolę LER, a które LSR, ich działanie jest poprawne i w pełni kompatybilne. Jest to bardzo istotna obserwacja, gdyż technice MPLS często zarzuca się, że jej działanie jest mocno zależne od wykorzystywanej platformy [8]. Przeprowadzony eksperyment pokazuje jednak, że swobodnie możemy ze sobą zestawiać rozwiązanie jednego z głównych producentów sprzętu sieciowego, jakim jest Cisco, z pomysłem całkowicie opartym na produktach udostępnianych jako wolne oprogramowanie. Wynika stąd, że ze zgodnością różnych wersji MPLS nie jest tak źle i architekci sieci, wykorzystując tę technologię, nie muszą ograniczać się do użycia sprzętu od jednego producenta.

[2] sourceforge.net/projects/mpls-linux/

4.2 Wydajność MPLS i OpenFlow

Jak to już zostało opisane w poprzednich rozdziałach, jedną z motywacji do wykorzystania protokołów MPLS i OpenFlow często jest konieczność dostarczania usług o gwarantowanej jakości (QoS). Powszechnie przyjęło się, że QoS jest określane na podstawie następujących parametrów:

- przepustowość sieci,
- zmienność opóźnienia (jitter),
- liczba pakietów traconych w czasie transmisji.

W założeniach, dzięki zaawansowanym mechanizmom inżynierii ruchu, które są dostępne przy zastosowaniu opisywanych technologii, powinna istnieć możliwość poprawy wydajności transmisji pakietów w stosunku do tego, co oferuje Internet Protocol. Rozdział ten dokładnie opisuje sposób, w jaki starałem się sprawdzić wpływ zastosowania MPLS oraz OpenFlow na parametry opisujące jakość działania sieci.

4.2.1 Konfiguracja

Wykorzystywana w przeprowadzanych testach wydajności sieć, podobnie jak poprzednio, składała się z pięciu węzłów, gdzie węzły brzegowe były zwykłymi komputerami PC wykorzystywanymi przez użytkownika końcowego, a trzy środkowe pełniły funkcję szkieletu infrastruktury sieciowej. Poglądowo dalej aktualny jest schemat przedstawiony na rysunku 6, ale należy zwrócić uwagę na zmiany w konfiguracji, szczegółowo opisane w dalszej części.

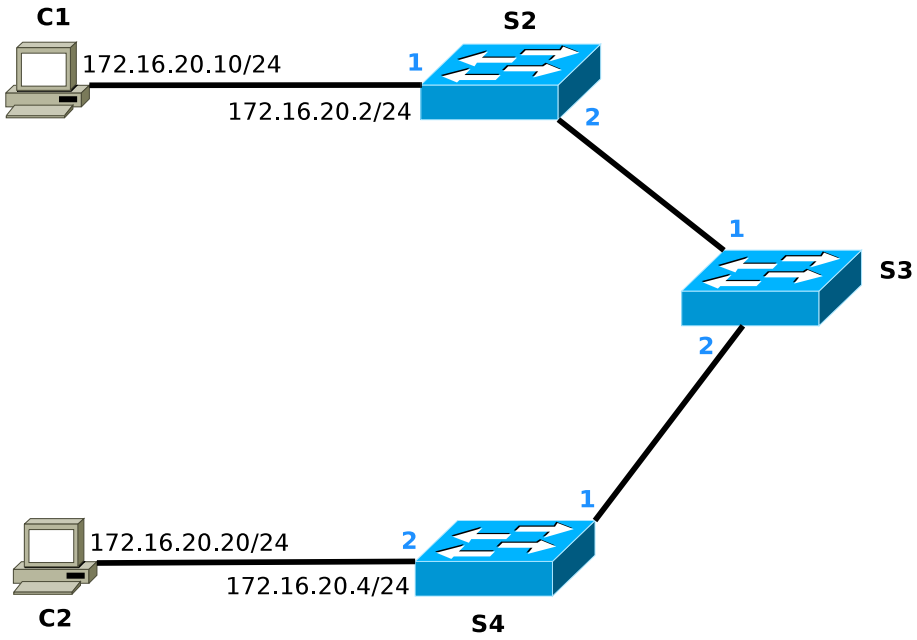
Punktem odniesienia stosowanym do określenia zmian wydajności i jakości transmisji przy wykorzystaniu MPLS i OpenFlow, była sieć oparta na standardowym protokole IP. Rolę routerów pełniły komputery PC z zainstalowanym systemem operacyjnym Linux i włączoną opcją IP forwarding. Tablice routingu we wszystkich węzłach sieci zostały wypełnione w sposób statyczny, adekwatnie do tego, co jest pokazane na rysunku 6.

Z konfiguracją bazową opisaną w poprzednim akapicie porównywane były trzy rozwiązania oparte na różnych technologiach:

- routery programowe wykorzystujące pakiet MPLS for Linux,
- routery sprzętowe Cisco obsługujące MPLS,
- przełączniki programowe wykorzystujące protokół OpenFlow.

Ustawienia urządzeń działających zarówno jako komputery użytkowników końcowych, jak i routery wykorzystujące MPLS, są dokładnie takie same, jak w przypadku testowania kompatybilności różnych implementacji tego rozwiązania, opisywanej w części 4.1, a przykładowe skrypty konfiguracyjne są dostępne w dodatku A (Listingi: A.1, A.2, A.3, A.4).

Trochę inaczej wygląda sytuacja, jeśli pod uwagę weźmiemy sytuację, w której używamy programowych przełączników wykorzystujących protokół OpenFlow. Sprzęt sieciowy działający w warstwie łącza danych modelu ISO OSI wymusza, aby węzły końcowe znajdowały się w obrębie tej samej podsieci. Przełączniki nie posługują się także routingiem opartym na przypisanych do interfejsów adresach IP, lecz przy przekazywaniu ruchu kierują się właściwymi nazwami portów. Komunikacja z kontrolerem poprzez bezpieczny kanał jest możliwa dzięki temu, że przełączniki nasłuchują na 6634. porcie TCP. Schematycznie sieć przedstawia rysunek 7. Przełączniki zostały opisane odpowiednio jako S2, S3, oraz S4, a znajdujące się przy nich liczby oznaczają numery ich portów. Jak można zauważyć, na schemacie (rys. 7) nie widać urządzenia pełniącego rolę kontrolera. Dla potrzeb tego eksperymentu wystarczało ograniczyć się do dodania dwóch prostych reguł do tablicy przepływów każdego z przełączników. Jest to podejście analogiczne do umieszczania statycznych wpisów w tablicy routingu czy tablicy przełączania etykiet MPLS. Do tego celu nie potrzeba żadnego wyrafinowanego, sterowanego programowo kontrolera zarządzającego ruchem w sieci i można ograniczyć się do użycia narzędzia o nazwie *dpctl*. Jest to program umożliwiający komunikację z przełącznikiem za pomocą protokołu OpenFlow. Dzięki niemu można dodawać i usuwać wpisy do tablic przepływów, oglądać statystyki ruchu i przeprowadzać podstawową konfigurację przełączników. Skrypty odtwarzające dokładną konfigurację wykorzystywaną w tym porównaniu są dostępne w dodatku A (Listingi: A.5, A.6, A.7).



Rysunek 7: Wydajność protokołu OpenFlow – schemat sieci

Badanie wydajności sieci zbudowanych w oparciu o różne technologie, odbywało się przy użyciu opisywanych wcześniej narzędzi do generowania ruchu (*mgen*, *iperf*). Programy te dają użytkownikowi możliwość obserwacji wybranych parametrów transmisji. Otrzymane rezultaty zostały szczegółowo omówione w kolejnym podrozdziale.

4.2.2 Wyniki

Przepustowość sieci

Przepustowość sieci była sprawdzana przy użyciu programu *iperf*. Jako punkt odniesienia przyjąłem przepustowość łącza, jaką obserwujemy w sytuacji, gdy komputery PC użytkowników końcowych są połączone bezpośrednio i pracują w obrębie jednej podsieci. Uśredniony wynik, jaki otrzymałem przy nominalnej szybkości łącza wynoszącej 100 Mb/s, to 94,53 Mb/s.

Kolejnym krokiem było zweryfikowanie, jak zmieni się ta wartość dla konfiguracji pokazanej na rysunku 6 i wykorzystaniu mechanizmu IP forwarding. Uśredniony wynik to dokładnie 94,5 Mb/s.

Technologia	**Przepustowość [Mb/s]**	**Odchylenie standardowe [Mb/s]**
Ip forwarding	94,5	0,071
Mpls for Linux	94,28	0,044
Cisco MPLS	94,2	0
OpenFlow	94,9	0

Tabela 1: Przepustowość sieci w zależności od zastosowanej technologii

Traktując powyższe wyniki jako pewnego rodzaju wzorzec, sprawdziłem, jak w odniesieniu do nich prezentuje się przepustowość sieci opartej kolejno na technologiach: MPLS for Linux, MPLS w implementacji Cisco oraz OpenFlow. Otrzymane uśrednione rezultaty są zaprezentowane w tabeli 1.

Przedstawione wyniki (tab. 1) pozwalają nam na stwierdzenie, że używanie dowolnej z badanych technologii nie ma większego wpływu na taki parametr sieci, jakim jest jej przepustowość. Widzimy, że wartości średniej arytmetycznej policzonej na podstawie czterech pomiarów wykonanych dla każdej z metod są bardzo do siebie zbliżone. Zakładając, że rezultaty otrzymane przy pomocy programu *iperf* są wiarygodne, możemy ograniczyć się do tak niewielkiej próby, jaką są tylko cztery pomiary, ponieważ odchylenie standardowe zmiennej losowej jest niewielkie. Pokazuje to także, że niezależnie od zastosowanej technologii sterowania ruchem, sieć działa w sposób przewidywalny, jeśli pod uwagę bierzemy jej przepustowość.

Zmienność opóźnienia (jitter)

Narzędzie *iperf* do pomiaru przepustowości łącza używa techniki opartej na protokole TCP, ale umożliwia ono również sprawdzenie parametrów sieci z wykorzystaniem UDP (ang. User Data Protocol). UDP, w przeciwieństwie do TCP, jest protokołem bezpołączeniowym. Nie dostarcza on mechanizmów potwierdzenia dotarcia pakietu do celu oraz retransmisji zgubionej porcji danych. Dodatkowo wiadomości mogą docierać do odbiorcy w nieuporządkowanej kolejności. Ma on jednak jedną bardzo ważną przewagę nad TCP – jest szybszy. Dzięki tej zalecie jest powszechnie stosowany przy transmi-

Technologia	**Jitter [ms]**	**Odchylenie standardowe [ms]**
Ip forwarding (x1)	0,016	0,006
Ip forwarding (x11)	7,00	1,453
Mpls for Linux (x1)	0,147	0,161
Mpls for Linux (x11)	7,6	2,161
Cisco MPLS (x1)	0,103	0,138
Cisco MPLS (x11)	6,43	2,000
OpenFlow (x1)	0,073	0,122
OpenFlow (x11)	7,58	1,058

Tabela 2: Zmienność opóźnienia (jitter) w zależności od zastosowanej technologii

sjach wideo typu „live streaming" oraz w usługach VoIP, gdzie minimalizacja opóźnień i jitteru, czyli parametru określającego nieregularność docierania danych do odbiorcy, jest kluczowym aspektem.

Wykorzystując dokładnie tę samą topologię sieci, której używałem przy poprzednich pomiarach, ale tym razem bazując na transmisji datagramów UDP, sprawdziłem, jak zmienia się wartość jitteru w zależności od zastosowanej technologii kierowania ruchem. Dodatkowo zbadałem, jaki wpływ na powyższy parametr ma przesyłanie danych z kilku różnych portów jednocześnie. Taki test może symulować rzeczywistą sytuację, gdy w kanale transmisyjnym spotykają się pakiety z różnych źródeł. Jako odnośnik, podobnie jak poprzednio, możemy traktować rezultat otrzymany w sytuacji, gdy dwa komputery były podłączone bezpośrednio w ramach jednej podsieci, a do komunikacji był używany jeden port na kliencie i jeden na serwerze. Średnia wartość obliczona na podstawie ośmiu prób to 0,017 ms przy odchyleniu standardowym równym 0,006 ms. Pozostałe wyniki zostały zaprezentowane w tabeli 2.

Dla każdej z badanych konfiguracji ośmiokrotnie przeprowadziłem identyczne pomiary, a następnie obliczyłem średnią arytmetyczną z uzyskanych wyników. Ma to na celu wyeliminowanie przypadkowości spośród otrzymanych rezultatów oraz dostarczenie powtarzalnych i wiarygodnych danych do

dalszej analizy. Dodatkowo prezentowane jest także odchylenie standardowe obserwowanej zmiennej losowej, które określa jak zróżnicowane były wyniki. Oznaczenia umieszczone w nawiasach: „x1" i „x11" mówią, w jakim trybie była przeprowadzana transmisja datagramów. Odpowiednio, świadczy to o tym, że przesył odbywał się pomiędzy jednym portem klienta i jednym portem serwera lub jedenastoma portami klienta i jednym portem serwera. Wartości jitteru, znajdujące się w tabeli 2 odnoszą się do pojedynczego zestawienia port-port.

Otrzymane wyniki nie pokazują wyraźnej przewagi żadnej z badanych technologii. Można stwierdzić, że wszystkie pracują tak samo dobrze, gdyż wartości jitteru nie są duże. Ciekawą obserwacją jest to, że zwiększenie liczby równoległych transmisji powoduje znaczny wzrost wartości jitteru. Dla IP forwarding i OpenFlow jest on ponad stukrotny przy tylko nieco ponad dziesięciokrotnym zwiększeniu liczby nadawców. Wynika to jednak prawdopodobnie z tego, że serwer przyjmuje „grupowo" datagramy nadawane z danego portu, co w konsekwencji powoduje nieregularność z jaką dane docierają do celu. Jako punkt odniesienia dla otrzymanych rezultatów możemy przyjąć stwierdzenie, że akceptowalna wartość jitteru dla transmisji obrazu wideo w sieciach pakietowych to 30 ms [12]. Przedstawionym wynikom sporo jeszcze do tego progu brakuje, ale pod uwagę należy wziąć niewielki rozmiar sieci, na której były przeprowadzane testy.

Utrata pakietów

Do zbadania, jak zmienia się ilość gubionych podczas transmisji danych w zależności od zastosowanej technologii, używałem programu *mgen*. Wprawdzie *iperf* również dostarcza statystyk dotyczących utraconych w czasie transmisji pakietów, ale *mgen* daje większe możliwości specyfikowania charakteru generowanego ruchu. Jest bardziej elastyczny pod względem doboru rozmiaru pakietu i charakterystyki przesyłu w czasie.

W tej serii testów, ruch w sieci był generowany poprzez przesyłanie datagramów UDP. Jakość transmisji była sprawdzana przy czterech różnych poziomach obciążenia sieci. Zmieniana była zarówno częstotliwość, z jaką

Technologia	**50B-med**	**100B-med**	**1000B-sat**	**100B-low**
Ip forwarding	599720/700164	371041/375007	112341/117628	117793/120007
Mpls for Linux	634410/741106	374788/375007	113348/117635	120005/120008
Cisco MPLS	611927/736478	373556/375007	117357/117633	120007/120007
OpenFlow	306829/752828	290801/373055	106118/117722	108209/120007

Tabela 3: Liczba pakietów poprawnie odebranych w stosunku do wszystkich wysłanych

pakiety były wysyłane, jak i ich rozmiar. Maszyny brzegowe pracowały w architekturze klient-serwer. Klient emitował informacje przez okres 10 sekund przy użyciu trzech portów UDP jednocześnie, natomiast serwer nasłuchiwał w tym czasie także na trzech różnych portach UDP. Przy określaniu charakterystyki ruchu użyto opcji „Periodic", co oznacza, że generowane były pakiety o stałym rozmiarze i w równych odstępach czasu. Pozostałe parametry zadane dla poszczególnych przypadków testowych są opisane poniżej, a otrzymane wyniki zaprezentowano na wykresie (rys. 8) oraz w tabeli 3. Dla każdego przypadku testowego przeprowadzono trzy próby, a przedstawione wyniki stanowią ich średnią arytmetyczną.

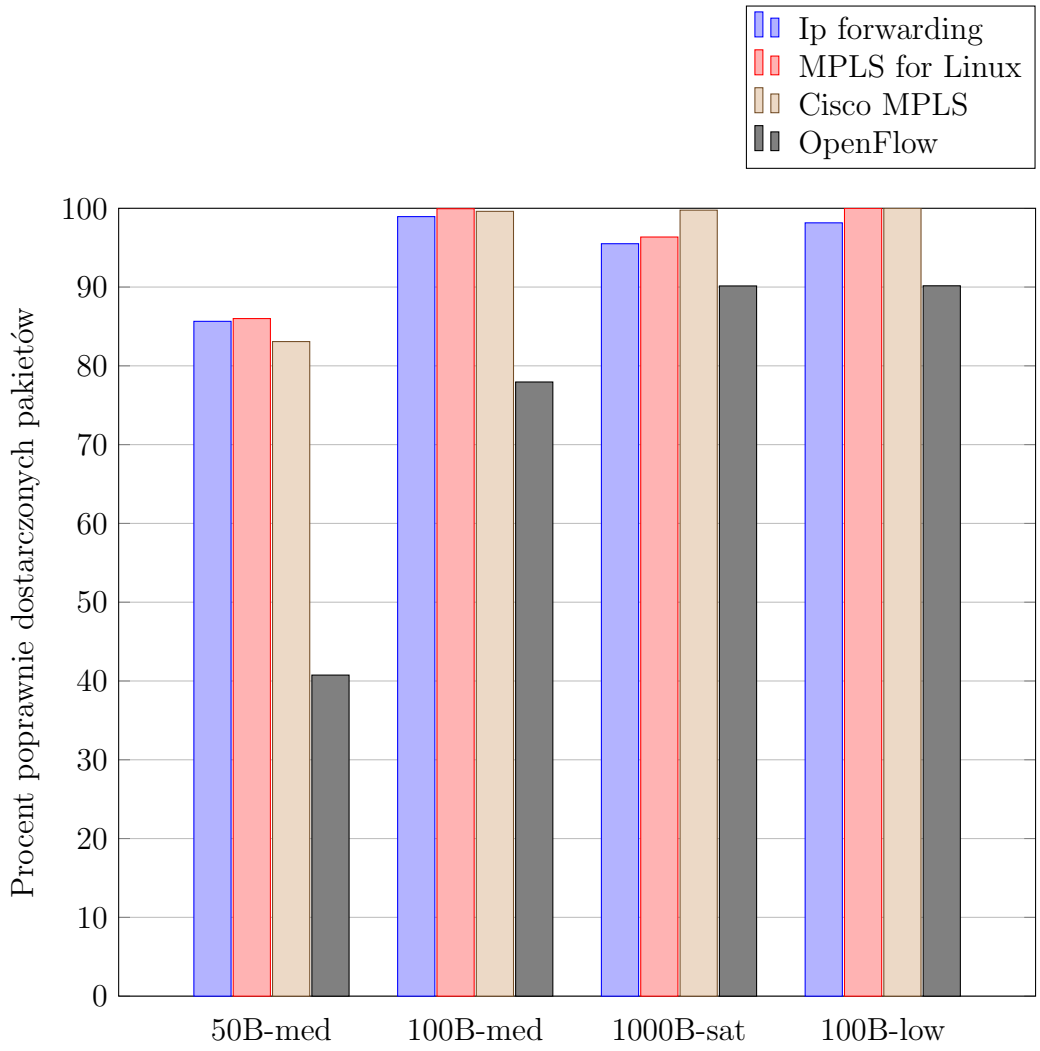
50B-med Wysyłano pakiety o rozmiarze 50 bajtów z częstotliwością 25000 razy na sekundę. Łącze nasycone w około 30 procentach.

100B-med Wysyłano pakiety o rozmiarze 100 bajtów z częstotliwością 12500 razy na sekundę. Łącze nasycone w około 30 procentach.

1000B-sat Wysyłano pakiety o rozmiarze 1000 bajtów z częstotliwością 4000 razy na sekundę. Łącze nasycone w około 96 procentach.

100B-low Wysyłano pakiety o rozmiarze 100 bajtów z częstotliwością 4000 razy na sekundę. Łącze nasycone w około 10 procentach.

Patrząc na otrzymane wyniki można stwierdzić, iż jakość pracy, mierzona jako procent poprawnie dostarczonych pakietów, wszystkich porównywanych technologii jest podobna. Trochę w tym zestawieniu odstaje sieć oparta na



Rysunek 8: Ilość poprawnie dostarczonych danych w zależności od zastosowanej technologii dla poszczególnych przypadków testowych

rozwiązaniu OpenFlow, co jest widoczne i potwierdzone dla każdego przypadku testowego. Tak podobne wyniki mogą być spowodowane doborem takich, a nie innych przypadków testowych. Wydawać by się mogło, że najbardziej selektywny test polegałby na transmisji możliwie małych pakietów z taką częstotliwością, aby obciążenie łącza oscylowało w okolicach stu procent. Problem polegał jednak na tym, że ani przy użyciu programu *mgen*, ani *iperf*, nie jesteśmy w stanie tego zrobić. Kłopotem jest ograniczona częstotliwość z jaką datagramy mogą być wysyłane. Maksymalna wartość tego parametru, jaka była używana w tym eksperymencie, to ta z pierwszego przypadku testowego, czyli 75000 razy na sekundę. Już przy tej wartości program ostrzegał, że dostarczone wyniki mogą być nieprecyzyjne. Widać to w danych zawartych w tabeli 3, gdzie w kolumnie „50B-med" liczba generowanych pakietów różni się w zależności od zastosowanej technologii. Ponadto trzeba pamiętać, że są to wartości średnie z trzech kolejnych pomiarów i o ile odchylenie standardowe otrzymywanych w kolejnych próbach wyników w trzech pozostałych przypadkach testowych było bardzo małe, to w tym było znaczące. Taka niedoskonałość używanych narzędzi spowodowała, że działanie zastosowanych protokołów było sprawdzane w warunkach znacznego nasycenia kanału (ok. 96%), ale wysyłane były wtedy duże porcje danych z niewielką częstotliwością, z czym, jak pokazują zebrane wyniki, wszystkie technologie radzą sobie całkiem dobrze.

Opóźnienie

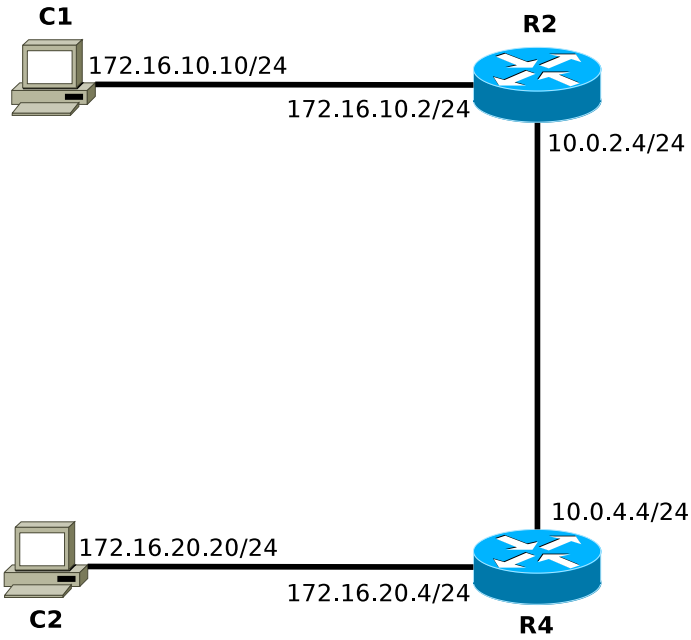
Opóźnienie, które w odniesieniu do sieci komputerowych często określane jest skrótem RTT (ang. Round Trip Time), jest to czas potrzebny na przesłanie sygnału od nadawcy do odbiorcy, a następnie z powrotem do nadawcy. Zwyczajowo parametr ten nie jest używany w stosunku do QoS, niemniej jednak może on służyć do porównania jakości rozwiązań, jakie uzyskujemy używając poszczególnych technologii.

Do pomiaru wartości RTT użyto programu *ping*. Podobnie, jak w poprzednich eksperymentach, wyniki otrzymane dla każdego przypadku testowego, są wartościami średnimi obliczonymi na podstawie kilku, w tym

Technologia	**Średnie opóźnienie [ms] dla pakietów o rozmiarze 78 B**	**Średnie opóźnienie [ms] dla pakietów o rozmiarze 50028 B**
Ip forwarding	0,235	9,574
Mpls for Linux	0,259	25,94
Cisco MPLS	0,259	44,499
OpenFlow	0,379	9,89

Tabela 4: Pomiar opóźnienia (RTT) w zależności od zastosowanej technologii

przypadku trzech, kolejnych pomiarów. Doświadczenie sprawdzające, jak na opóźnienie wpływa wybór technologii, polegało na przesyłaniu 10000 pakietów icmp o rozmiarze, w zależności od testu, 78 bajtów lub 50028 bajtów. Program *ping* pracował z włączoną opcją „flood", która powoduje, że kolejny pakiet jest wysyłany po powrocie poprzedniego lub, że pakiety są wysyłane z częstotliwością 100 razy na sekundę (zależnie od tego, w której sytuacji generowanych jest więcej pakietów). W tabeli 4 przedstawione są otrzymane wyniki. Rezultaty uzyskane w wyniku przesyłania pakietów o rozmiarze 78 bajtów są bardzo zbliżone do siebie, niezależnie od tego jaką technikę sterowania przepływem danych w sieci wybraliśmy. Przyczyną tego jest fakt, że podczas tych testów obciążenie infrastruktury było niewielkie i oscylowało w okolicach 1,3 %. Jedyne, co można zrobić za pomocą programu *ping*, aby wygenerować większy ruch, to zwiększyć rozmiar przesyłanych pakietów. Podczas tego eksperymentu używano pakietów o rozmiarze 50028 bajtów. W tym przypadku mieliśmy do czynienia z transmisją na poziomie 40 Mb/s. Załączone rezultaty pokazują, że w takiej sytuacji najmniejsze opóźnienie wprowadza rozwiązanie oparte na przekazywaniu IP. Bardzo zbliżone opóźnienie uzyskujemy stosując protokół OpenFlow. Zdecydowanie słabiej na tym tle wypada protokół MPLS, zwłaszcza w implementacji Cisco. Świadczy o tym zdecydowanie największe opóźnienie, a także to, że strata pakietów przy zastosowaniu rozwiązania firmy Cisco dochodzi do 98 %! Watro w tym miejscu zaznaczyć, że pozostałe technologie zapewniają, że do celu docierają wszystkie pakiety lub ich utrata jest rzędu setnych części procenta. Jeśli pod



Rysunek 9: Schemat sieci – cztery węzły

uwagę weźmiemy stosunek poprawnie odebranych danych do wszystkich wygenerowanych w sytuacji, gdy przesyłane były pakiety o rozmiarze 50 bajtów, to niezależnie od zastosowanej technologii wynosił on 100 %.

Postawioną na początku tej pracy hipotezą było to, że przełączanie etykiet MPLS powinno być szybsze od routingu opartego na nagłówkach IP. Przedstawione wyniki temu przeczą. Należy się jednak zastanowić, co może być tego przyczyną. Sieć wykorzystywana do testów składała się tylko z pięciu węzłów, natomiast domena MPLS to tylko trzy węzły: dwa LER i jeden LSR. Na urządzeniach LER ma miejsce dodawanie i usuwanie etykiety MPLS. Jest to operacja dość kosztowna czasowo. Korzyści z zastosowania MPLS powinny się ujawniać podczas przepływu pakietów przez węzły LSR, gdzie ma miejsce przełączanie etykiet MPLS. Aby sprawdzić czy tak sformułowana teza znajduje odzwierciedlenie w rzeczywistości, przeprowadziłem jeszcze jeden eksperyment. Skonfigurowałem sieć, działającą w oparciu o MPLS for Linux, składającą się z czterech węzłów, którą schematycznie przedstawia rysunek 9. Ograniczając się już tylko do rozpatrywania przypadku, kiedy przesyłane są pakiety o rozmiarze 50028 bajtów, otrzymujemy średnie opóźnienie równe

25,576 ms. Porównując to z wartością otrzymaną dla sieci składającej się z pięciu węzłów (rys. 6), narzut czasowy, jaki powstaje w momencie dołożenia urządzenia pełniącego rolę LSR wynosi tylko 0,364 ms. Pokazuje to, że sama operacja przełączania etykiet MPLS jest dużo bardziej efektywna niż routing oparty na nagłówkach IP. W rzeczywistych sieciach operatorskich mamy zwykle do czynienia ze znacznie dłuższymi ścieżkami przepływu oraz z całkowicie innym stosunkiem węzłów LSR do węzłów LER. W takich warunkach znacznie częściej ma miejsce operacja przełączania etykiet niż ich dodawania bądź usuwania, co powoduje uwydatnienie się korzyści płynących z zastosowania MPLS. Niestety pokazanie tego zjawiska nie jest do końca możliwe w przygotowanym środowisku laboratoryjnym co wynika, z ograniczeń dostępnego sprzętu.

Niedoskonałości protokołu IP – przeglądanie dużej tablicy routingu

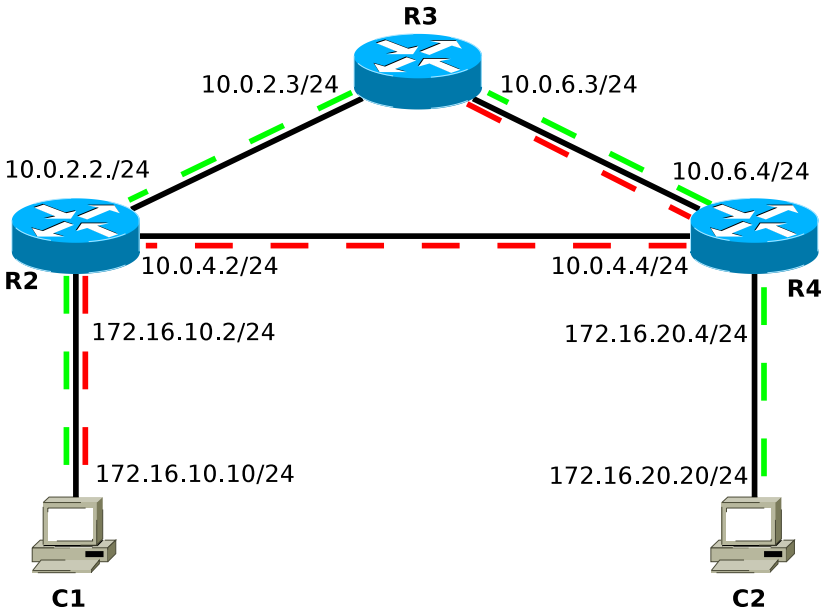
Celem poprzednich eksperymentów było zbadanie jakości działania technologii MPLS i OpenFlow w odniesieni do statycznego routingu IP. Jednym z powodów rozwijania tych protokołów jest chęć zmniejszenia wpływu słabych stron protokołu IP na jakość działania sieci komputerowych. Przeprowadzone testy pokazały jednak, że protokół IP radzi sobie w stworzonych warunkach bardzo dobrze i ciężko wskazać jego słabe punkty, co może z kolei świadczyć o tym, że zaproponowane testy nie były zbyt selektywne. Stąd powstał pomysł, aby stosując konfigurację sieci opartą na routingu IP (rys. 6), dodać do tablicy routingu urządzenia R3 dużą liczbę losowych adresów sieci o długich maskach. Za pomocą skryptu zostało wygenerowane 10000 tysięcy takich wpisów. Do sprawdzenia, jak wpłynie to na jakość pracy sieci użyto programu *mgen*, który był uruchamiany z takimi samymi parametrami, jak podczas testu „50B-med" opisanego w części „Utrata pakietów". Niestety otrzymane wyniki wciąż nie były satysfakcjonujące. Procent poprawnie transmitowanych datagramów w przypadku, gdy w tablicy routingu nie było żadnych „sztucznych" wpisów oscylował w okolicy 80 %, natomiast w sytuacji, gdy mieliśmy do czynienia z dużą tablicą routingu, wynosił on około 74 %. Tak niewielki spadek jakości transmisji, mógł być spowodowany tym,

że router R3 przechowywał odpowiednie wpisy w pamięci podręcznej. Aby wyłączyć tę opcję systemu Linux posłużyłem się poleceniem
 # echo 3000000000 > /proc/sys/net/ipv4/rt_cache_rebuild_count,
które powoduje wyłączenie opcji przechowywania w cache'u ostatnio używanych wpisów z tablicy routingu. Niestety również i ten zabieg nie przyniósł spodziewanych efektów. Sieć w dalszym ciągu pracowała bardzo dobrze, o czym świadczą następujące rezultaty. Liczba poprawnie przesłanych danych, w czasie testu, gdy w tablicy routingu urządzenia R3 nie było dodatkowych wpisów to średnio około 85 %, a w sytuacji, gdy we wspomnianej tablicy znajdowało się 10000 dodatkowych wpisów to 75 %.

Zastosowane zabiegi znów nie uwydatniły słabości rozwiązania opartego na routing IP. Biorąc pod uwagę niedokładność działania programu *mgen*, otrzymane różnice w liczbie poprawnie odebranych pakietów są zbyt małe, aby móc mówić o sytuacji, w której to sieć działa słabo, a my próbujemy poprawić jej wydajność stosując czy to MPLS, czy OpenFlow. Prowadzi to do stwierdzenia, że dysponując wykorzystywanymi narzędziami nie jesteśmy w stanie zaprojektować i zrealizować testu wydajności, na tyle wymagającego, aby w pełni uwypuklić niedoskonałości protokołu IP.

4.3 MPLS i OpenFlow jako narzędzia do inżynierii ruchu

Dynamiczny rozwój Internetu na przestrzeni ostatnich lat spowodował, że bardzo istotnym zagadnieniem stało się efektywne wykorzystywanie infrastruktury sieciowej. Wiąże się z tym pojęcie inżynierii ruchu, które opisowo oznacza wszelkie działania mające na celu ocenę i optymalizację wydajności sieci komputerowych. Świadczenie za pomocą technologii internetowych usług nowego rodzaju takich jak transmisja obrazu czy dźwięku w czasie rzeczywistym albo interaktywne gry w trybie online, doprowadziły do zdefiniowania terminu Quality of Service. To właśnie sprostanie wymaganiom klientów jest jedną z głównych przyczyn rozwoju inżynierii ruchu. Problemem jest jednak to, że protokół IP, na którym oparta jest sieć Internet, udo-

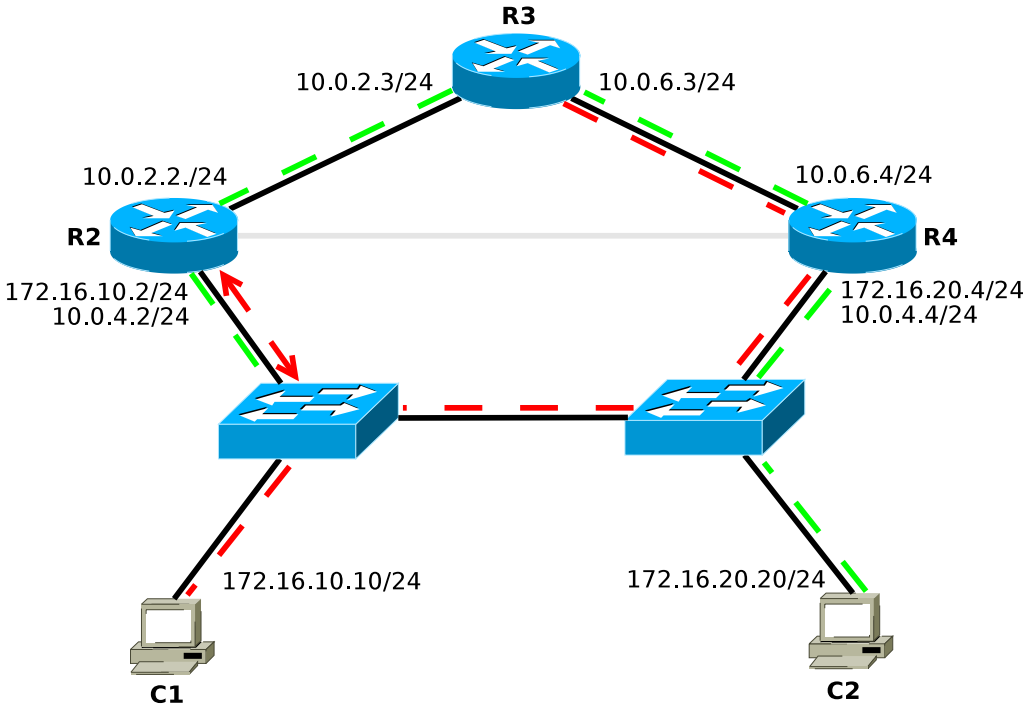


Rysunek 10: Traffic engineering – MPLS for Linux

stępnia bardzo ubogi zestaw narzędzi do efektywnego sterowania ruchem. Jednym z powodów, dla których takie technologie jak MPLS czy OpenFlow są opracowywane i rozwijane, jest właśnie poszerzenie możliwości w tym zakresie. W tej części mojej pracy pokażę, jak za ich pomocą można zdefiniować nietrywialne reguły określające przepływ ruchu w sieci.

4.3.1 Założenia i konfiguracja

W eksperymencie tym do pokazania możliwości, jakie pojawiają się w zakresie inżynierii ruchu, gdy korzystamy z technologii MPLS lub OpenFlow, używałem trzech różnych konfiguracji środowiska, stanowiących namiastkę rzeczywistych sieci. Podyktowane to było zarówno ograniczeniami sprzętu dostępnego w laboratorium jak i zróżnicowanymi możliwościami wspomnianych protokołów. Główny cel oraz efekty są jednak we wszystkich przypadkach takie same. Chodziło o to, aby ruch do jednego punktu docelowego kierować po różnych trasach w zależności od jego źródła pochodzenia. Wykorzystując komputery znajdujące się w laboratorium i technologię MPLS for Linux, przygotowałem konfigurację, której schemat znajduje się na rysunku 10.

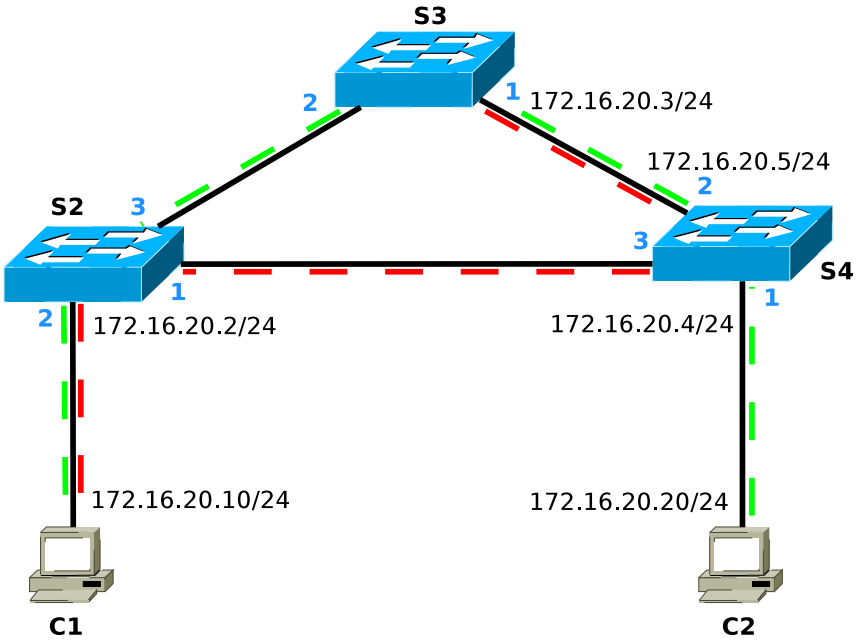


Rysunek 11: Traffic engineering – Cisco MPLS

Układ ten dokładnie spełnia wszystkie założenia, które zostały postawione przed przystąpieniem do przygotowywania tego testu. Węzły pełniące rolę komputerów użytkowników końcowych pracują w dwóch różnych podsieciach. Do zapewnienia żądanej funkcjonalności nie są wykorzystywane żadne dodatkowe urządzenia. Trochę inaczej sytuacja wygląda, gdy to samo chcemy uzyskać wykorzystując czy to protokół MPLS w implementacji Cisco, czy OpenFlow.

Aby zrealizować postawione założenia przy użyciu routerów i implementacji protokołu MPLS firmy Cisco, przygotowałem środowisko, którego schemat jest przedstawiony na rysunku 11.

Do stworzenia konfiguracji sieci, jaka była mi potrzebna do wykonania tego zadania, musiałem wykorzystać dwa dodatkowe przełączniki dostępne w laboratorium. Spowodowane to było faktem, że każdy z dostępnych routerów Cisco dysponuje tylko dwoma portami FastEthernet. System IOS daje jednak możliwość przypisywania do interfejsu więcej niż jednego adresu IP,



Rysunek 12: Traffic engineering – OpenFlow

dlatego przy użyciu wspomnianych przełączników udało się osiągnąć pożądaną funkcjonalność.

Jeszcze inaczej sytuacja wyglądała, gdy w celu osiągnięcia podobnego efektu, zastosowałem technologię OpenFlow. Urządzenia pełniące role przełączników programowych to komputery PC z odpowiednim zestawem narzędzi i wystarczającą liczbą kart sieciowych. Nie trzeba więc było stosować żadnych dodatkowych przyrządów ani innych wymuszonych rozwiązań. Należało jednak uwzględnić fakt, że przełączniki programowane przy pomocy protokołu OpenFlow pracują w warstwie łącza danych. Stąd wymóg, aby komunikujące się węzły znajdowały się w obrębie jednej podsieci. Schemat sieci jest pokazany na rysunku 12. Niebieską czcionką opisano poszczególne numery portów przełączników.

W omawianej konfiguracji uwagę mogą zwracać adresy IP przypisane do portu 1 urządzenia S3 oraz portu 2 urządzenia S4. Nie przeczą one temu, o czym wspomniałem wcześniej, że przełączniki OpenFlow pracują w drugiej warstwie modelu ISO OSI. Adresacja ta nie jest wykorzystywana przez przełączniki do sterowania przepływem ruchu. Węzeł S3 podczas tego eks-

perymentu pełni jednocześnie rolę komputera użytkownika końcowego, jak i programowego przełącznika OpenFlow. Port 2 maszyny S4 pełni w takim przypadku rolę interfejsu brzegowego domeny OpenFlow. Takie rozwiązanie zostało zastosowane, ponieważ jest ono analogiczne do ustawień stosowanych dla protokołu MPLS, a jednocześnie ogranicza liczbę wykorzystywanych fizycznych urządzeń, nie wpływając przy tym na otrzymane rezultaty.

Skrypty uruchamiające programowe przełączniki OpenFlow i konfigurujące je zgodnie z wymaganiami opisywanego testu są zawarte w dodatku A (Listingi: A.8, A.9, A.10). Znajdują się tam także sekwencje poleceń potrzebne do ustawienia prawidłowego działania routerów Cisco (Listingi: A.11, A.12, A.13), a także skrypty przygotowujące komputery do pełnienia roli programowych routerów używających MPLS for Linux (Listingi: A.14, A.15, A.16). Tryb, w jakim pracowały komputery C1 i C2, jest taki sam jak w adekwatnych, opisanych wcześniej, eksperymentach.

4.3.2 Wyniki

Prezentowany w tym teście przykład zastosowania bardziej zaawansowanego sterowania ruchem przy wykorzystaniu funkcji protokołów MPLS i OpenFlow polegał na przesyłaniu informacji do odbiorcy, którym był węzeł C1, po różnych trasach, w zależności od tego, kto był ich nadawcą. Rolę węzłów źródłowych pełniły maszyny: C2 i, w zależności od technologii, R3 lub S3. Ruch na trasie od C2 do C1 był transmitowany wzdłuż trasy oznaczonej zieloną, przerywaną linią, a pomiędzy R3 (lub S3) i C1 – czerwoną (rys. 10, rys. 11, rys. 12).

Konfiguracja taka może być podyktowana tym, że łącze pomiędzy punktami R3 a R4 (lub S3 a S4) charakteryzuje się większą przepustowością niż odcinki R2 – R3 (lub S2 – S3) oraz R2 – R4 (lub S2 – S4). W takim przypadku dostawca usług może starać się poprawić jakość dostarczanego sygnału poprzez rozłożenie obciążenia uwzględniające wąskie gardła wykorzystywanej infrastruktury.

Najprostszą możliwością sprawdzenia poprawności zastosowanej konfiguracji było uruchomienie jednocześnie transmisji na trasach R3 – C1 (lub S3 –

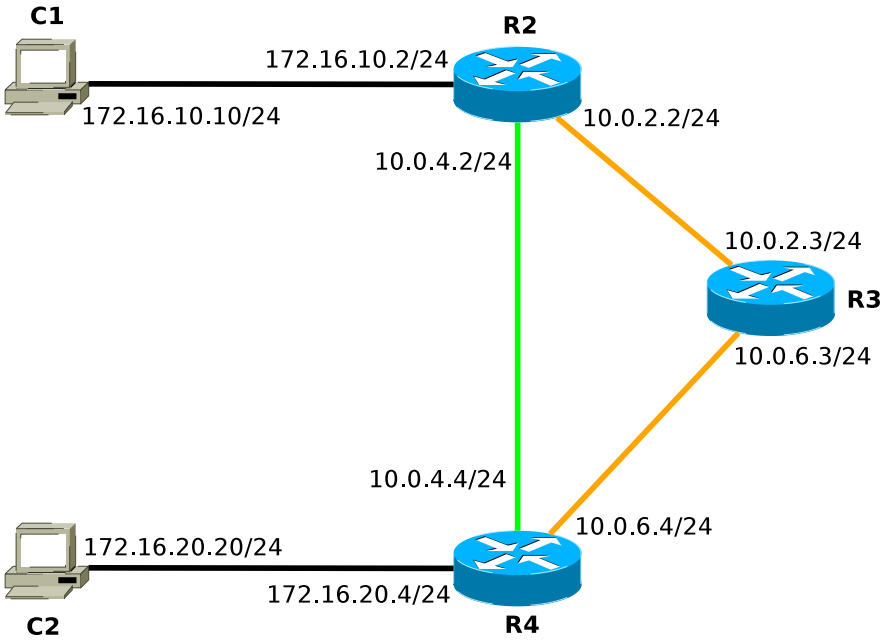
C1) oraz C2 – C1, a następnie przerwanie łącza między R2 – R3 (lub S2 – S3) czy R2 – R4 (lub S2 – S4). W tym eksperymencie do generowania ruchu wystarcza narzędzie *ping*. W momencie rozłączenia dowolnego z wymienionych powyżej odcinków transmisja na odpowiedniej trasie zostaje przerwana. Jednocześnie obserwujemy, że nie ma to żadnego wpływu na drugi, uruchomiony równolegle, przepływ. W ten prosty sposób pokazujemy, że nasza rzeczywista konfiguracja działa tak, jak założyliśmy.

4.4 „Failover scenario" – niezawodność sieci dzięki protokołowi MPLS

Niezawodność dostarczania usług jest w dzisiejszym świecie bardzo, jeśli nawet nie najbardziej, istotnym aspektem oceny jakości sieci komputerowych. Praca ludzi, właściwie chyba już we wszystkich branżach, jest w mniejszym lub większym stopniu oparta na złożonych systemach informatycznych wykorzystujących łączność sieciową. Każdy, nawet chwilowy, przestój spowodowany awarią łącza, może generować duże straty dla przedsiębiorstwa. Aby tego uniknąć, projektując sieci, często tworzy się łącza zapasowe dla kluczowych, z biznesowego punktu widzenia, ich fragmentów. Ta część pracy zawiera przegląd możliwości, jakie dają nam omawiane technologie w zakresie zabezpieczenia naszej sieci przed awarią łącza.

4.4.1 Założenia i konfiguracja

Konfiguracja sieci zestawiona w laboratorium na potrzeby testów technologii MPLS for Linux jest schematycznie pokazana na rysunku (rys. 13). Chcąc pokazać możliwości rozwiązania dostarczanego przez produkty firmy Cisco musiałem dokonać drobnych zmian w przygotowanym środowisku. Spowodowane było to ograniczeniami urządzeń dostępnych w laboratorium. Każdy ze znajdujących się tam routerów ma tylko dwa porty FastEthernet. Na szczęście dzięki opcjom dostępnym w systemie IOS i użyciu dodatkowego przełącznika udało się uzyskać konfigurację (rys. 14), przy której możliwe

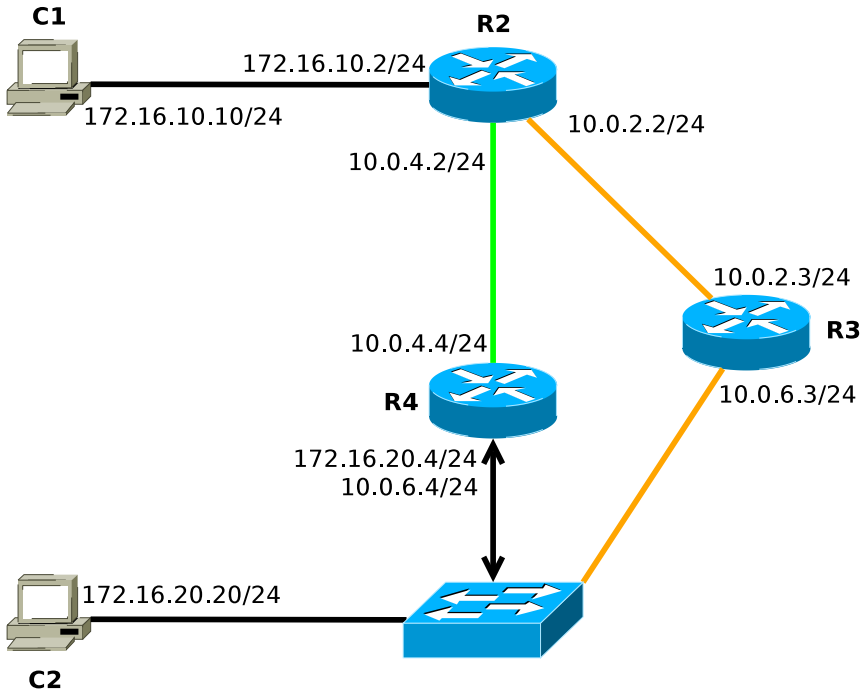


Rysunek 13: Failover scenario – MPLS for Linux

jest zademonstrowanie opcji zabezpieczających przed dotkliwymi skutkami awarii łącza.

Rozwiązanie minimalizujące odczuwalne dla odbiorców końcowych efekty awarii łącza, jakie jest dostępne przy użyciu MPLS for Linux, polega na okresowym badaniu stanu kanału transmisyjnego i dostosowywaniu kształtu specjalnych tablic routingu oraz tablic przełączania etykiet do aktualnej formy sieci. Do określania czy połączenie między węzłami działa prawidłowo, czy nie, wykorzystałem narzędzie *ethtool*. Jest to prosty program, który może być używany do sprawdzania stanu interfejsów sieciowych. Jedną z informacji, którą można za jego pomocą uzyskać, jest to czy połączenie na określonym interfejsie działa poprawnie.

W wykorzystywanym przykładzie (rys. 13) łącze główne pomiędzy komputerami użytkowników końcowych zaznaczono kolorem zielonym, a łącze zastępcze kolorem pomarańczowym. Domyślnie komunikacja odbywa się z wykorzystaniem łącza głównego, a w przypadku jego awarii używana jest trasa zapasowa. Monitorowanie podstawowego kanału transmisyjnego wykonywane jest poprzez uruchomiony w tle skrypt, który w zależności od wyniku do-



Rysunek 14: Failover scenario – Cisco MPLS

starczonego przez *ethtool* usuwa i dodaje wpisy w zmodyfikowanej tablicy routingu. Po stwierdzeniu awarii głównej arterii, pakiety są kierowane na ścieżkę zastępczą. Stan podstawowej trasy jest jednak wciąż monitorowany i w momencie, gdy znów stanie się drożna, nastąpi kolejna zmiana trasowania, a ruch będzie odbywał się po głównym łączu. Przykładowy skrypt konfiguracyjny dostępny jest dodatku A (Listing: A.17).

Z jeszcze ciekawszym pomysłem mamy do czynienia, gdy korzystamy z routerów Cisco. Możemy jednocześnie zdefiniować kilka ścieżek łączących węzły brzegowe i określić ich wagi. W ten sposób możemy zdefiniować główną i zapasową trasę przepływu danych. W zastosowanym przeze mnie rozwiązaniu współpracują routery programowe i sprzętowe. Węzeły R2 i R3 działają dokładnie tak samo jak w przykładzie pokazującym możliwości technologii MPLS for Linux, a węzeł R4 jest teraz routerem sprzętowym. Podobnie jak poprzednio, na schemacie (rys. 14) kolorem zielonym zaznaczono główną drogę służącą do transmisji danych pomiędzy C1 i C2, a pomarańczowym – zastępczą. Router Cisco dysponuje mechanizmem automatycznego wykrywa-

nia stanu interfejsów i w przypadku awarii łącza, na przykład zestawiającego bezpośrednio urządzenia R2 i R4, przełącza trasę i rozpoczyna transmisję pakietów po, uprzednio zdefiniowanej, ścieżce zapasowej. Gdy system stwierdzi, że łączność została przywrócona ruch ponownie zostanie skierowany na krótszą ze ścieżek. Szczegółowa konfiguracja routera sprzętowego R4 także jest dostępna w dodatku A (Listing: A.18).

4.4.2 Wyniki

Scenariusz testowy pozwalający sprawdzić czy stworzona konfiguracja zapewnia działanie zgodne z założeniami, polegał na symulowaniu awarii łącza głównego poprzez usunięcie przewodu łączącego węzły R2 i R4. Sprawdziłem doświadczalnie, że w takiej sytuacji, mechanizmy opisane w części 4.4.1, działają zgodnie z założeniami, co stanowi o poprawności zastosowanego rozwiązania. Czas, w jakim rozwiązanie oparte na technologii MPLS for Linux reaguje na uszkodzenia sieci, a także na ewentualne przywrócenie pełnej jej funkcjonalności jest uzależniony od wartości parametru skryptu, który, uruchomiony w tle, monitoruje stan sieci. Dobór tej wartości musi być kompromisem pomiędzy dostępnymi zasobami, a wymaganiami jakościowymi, jakie ma spełniać tworzona przez nas architektura. W przypadku urządzeń Cisco sprawdzanie stanu łącza leży całkowicie w gestii systemu IOS. Wyjątkowo sprawnie zachodzi zmiana trasy przesyłania pakietów, gdy łącze główne jest przywracane do stanu sprzed awarii. Znacznie dłużej trwa zdiagnozowanie, że kanał transmisyjny nie pracuje prawidłowo. Dodatkowo w eksperymencie tym wykorzystano kompatybilność otwartej i komercyjnej implementacji protokołu MPLS oraz pokazano jak przy takiej konfiguracji, sieć radzi sobie w sytuacji kryzysowej. Reasumując, eksperyment ten demonstruje jak wykorzystując technologię MPLS i mechanizmy służące do detekcji stanu łącza, skutecznie skonfigurować trasę zastępczą.

Rozdział 5

Podsumowanie

Ostatni rozdział pracy zawiera podsumowanie przeprowadzonych prac oraz prezentuje płynące z nich wnioski w odniesieniu do założeń i tezy sformułowanej w początkowej części pracy. Przedstawia on także refleksje na temat problemów, które pojawiły się w trakcie przeprowadzanych eksperymentów oraz pokazuje, w jaki sposób można by ewentualnie kontynuować pracę na badanym tematem.

5.1 Przeprowadzone prace

Przeprowadzone prace można podzielić na dwa zasadnicze etapy, które kolejno zostały opisane w rozdziałach: 3 i 4. Zanim mogłem przejść do właściwego tematu pracy, czyli badania efektywności protokołów MPLS i OpenFlow, musiałem przygotować środowisko, w którym miały być przeprowadzane testy wydajności. Ten pierwszy etap okazał się być równie pracochłonny jak główna część pracy. Przyczyną tego była potrzeba konfiguracji środowiska działającego w aż trzech różnych technologiach, dodatkowo przeprowadzana na fizycznym sprzęcie, który też czasami odmawiał współpracy. Poruszane zagadnienia był dla mnie całkowicie nowym tematem i to też na pewno miało wpływ na ilość czasu poświęconą na doprowadzenie dostępnej infrastruktury do stanu, który umożliwiał przeprowadzenie zaplanowanych testów.

Porównanie wydajności opisywanych technologii składało się z zaplanowania i przeprowadzenia szeregu testów oraz zebrania usystematyzowanych wyników, z których tylko cześć została zaprezentowana w pracy. Przygotowaną infrastrukturę wykorzystano do zweryfikowania tezy postawionej na wstępie, czyli sprawdzenia czy stosowanie OpenFlow i MPLS rzeczywiście pomaga w efektywniejszym wykorzystaniu zasobów sieciowych. Warto w tym miejscu zaznaczyć, że cały proces miał charaktery iteracyjny. Po wykonaniu serii testów i wstępnym przeanalizowaniu otrzymanych wyników, często miała miejsce modyfikacja przypadku testowego tak, aby ostatecznie prezentowane rezultaty był jak najbardziej wartościowe i wiarygodne.

Dodatkowo badane technologie są rozwijane także z myślą o dostarczeniu nowych możliwości w zakresie inżynierii ruchu. Przykładowe ich wykorzystanie w tym aspekcie zostało zademonstrowane w częściach: 4.3 i 4.4.

5.2 Wnioski

Celem tej pracy dyplomowej było zweryfikowanie prawdziwość stwierdzenia, że używając protokołów OpenFlow i MPLS jesteśmy w stanie efektywniej zarządzać ruchem w sieci komputerowej niż ma to miejsce w sytuacji stosowania najpowszechniejszego obecnie rozwiązania, czyli protokołu IP. Cząstkowe wnioski wynikające z poszczególnych eksperymentów były zazwyczaj prezentowane na bieżąco w częściach poświęconych na opis otrzymanych wyników. W tym rozdziale zaprezentuje zbiorcze zestawienie konkluzji, wynikających z przeprowadzonych prac.

Pierwszym istotnym stwierdzeniem, wynikającym z mojej pracy, jest fakt zgodności dwóch różnych implementacji protokołu MPLS. Sprawdziłem, że istnieje możliwość dostarczania usług internetowych za pomocą heterogenicznej sieci, wykorzystującej zarówno routery programowe działające w oparciu o MPLS for Linux, jak i routery sprzętowe Cisco.

Jeśli chodzi o główny aspekt pracy, czyli wydajność technologii OpenFlow i MPLS, to otrzymane rezultaty nie są do końca satysfakcjonujące. Złożyło się na to kilka czynników. Najpoważniejszym problemem okazała się niska jakość używanych programowych generatorów ruchu. Zarówno przy zastosowaniu

programu *mgen*, jak i *iperf* nie byłem w stanie sprokurować sytuacji, w której wykorzystywana infrastruktura sieciowa pracowałaby na granicach swoich możliwości. Tak jak już to zostało napisane wcześniej, najprawdopodobniej najbardziej selektywnym testem byłaby próba transmisji bardzo małych pakietów, które wysycałyby prawie w całości łącze o nominalnej przepustowości wynoszącej 100 Mb/s. Z pomocą mógłby tutaj przyjść sprzętowy generator ruchu, ale nie miałem możliwości wykorzystania takiego narzędzia w czasie swoich prac. Drugim problemem, który jest ściśle powiązany z pierwszym, była niemożność pokazania niedoskonałości wydajnościowych, architektury opartej na protokole IP. Założeniem było zademonstrowanie takiej sytuacji, a następnie pokazanie na tym tle, jakie nowe możliwości są dostarczane przez OpenFlow i MPLS. Niestety nie udało mi się zasymulować sytuacji, w której protokół i routing IP działałby wyraźnie źle. Trzecim, ale już nie tak istotnym problemem, były rozmiary przygotowanego środowiska testowego. Pięć węzłów to trochę za mało, aby móc w pełni pokazać zalety płynące z zastosowania opisywanych technologii. Wszystkie te przeciwności sprawiają, że wyciągniecie jednoznacznych wniosków na temat wydajności sieci opartej na OpenFlow czy MPLS jest bardzo trudne. Przeprowadzone testy porównujące parametry transmisji (przepustowość, stratę pakietów, opóźnienie czy zmienność opóźnienia) w zależności od zastosowanej technologii nie dostarczyły satysfakcjonujących rezultatów. Pokazują one jedynie, że w zadanych warunkach, właściwie wszystkie te rozwiązania pracują tak samo dobrze, a jakość świadczonych usług będzie w każdym przypadku dobra i porównywalna. Wynika to stąd, że różnice pomiędzy otrzymanymi rezultatami są naprawdę minimalne. Bardzo ciekawej obserwacji na temat szybkości przełączania etykiet MPLS dostarcza jednak jeden z testów opisanych w części 4.2. Przedstawione tam wyniki pokazują, że operacja zamiany etykiety na urządzeniach LSR jest dużo szybsza niż przeglądanie tablicy routingu przy podejmowaniu decyzji o przesłaniu pakietu do kolejnego węzła na trasie. Całkowite opóźnienie przy zastosowaniu technologii MPLS for Linux nie jest jednak mniejsze niż w przypadku używania IP forwarding, gdyż czas zyskiwany przy przejściu przez węzły LSR jest tracony na węzłach LER. Operacja dodawania i zdejmowania etykiety MPLS trwa dłużej niż wybór trasy na podstawie tablicy

routingu. Dysponując środowiskiem testowym składającym się z większej liczby węzłów można by lepiej uwydatnić tę zaletę protokołu MPLS. Jednak z uwagi na ograniczenia sprzętu dostępnego w laboratorium było to niemożliwe.

Odchodząc od aspektu szybkości działania OpenFlow i MPLS, w swojej pracy przedstawiłem przykłady wykorzystania tych protokołów w zakresie inżynierii ruchu. Pierwszym zagadnieniem było transmisja danych po arbitralnie dobranych trasach. Założenie było takie, żeby ścieżki przepływu prowadzące do jednego punktu docelowego uzależnić od źródła generującego ruch, czyli, żeby zastosować „routing źródłowy". Eksperyment szczegółowo opisany w części 4.3 pokazał, że za pomocą każdej z trzech badanych technologii jesteśmy w stanie uzyskać oczekiwane rezultaty.

Ostatnim tematem poruszonym w mojej pracy była kwestia zabezpieczenia sieci komputerowej przed skutkami nagłej awarii łącza. W części 4.4 przedstawiono, jakie możliwości w tym względzie daje nam MPLS for Linux, a jakie wersja MPLS stworzona przez firmę Cisco. Oba przedstawione rozwiązania działają skutecznie, ale lepszym wydaje się być to uzyskane przy zastosowaniu routerów Cisco, gdyż detekcja awarii odbywa się tam na poziomie systemu operacyjnego. Od administratora wymagane jest tylko odpowiednie skonfigurowanie trasy zapasowej, na którą automatycznie zostanie przekierowany ruch w sytuacji kryzysowej. Używając routerów programowych musimy dodatkowo używać narzędzi nadzorujących stan łącza i na przykład za pomocą skryptu uruchomionego w tle, właściwie reagować na zaistniałą sytuację. Rozwiązanie Cisco jest jednak bardziej naturalne i powinno zużywać mniej zasobów.

5.3 Kontynuacja prac

Biorąc pod uwagę otrzymane rezultaty dotyczące szybkości działania protokołów OpenFlow i MPLS na tle protokołu IP, to pozostawiają one pewien niedosyt. Przyczyna tej sytuacji została wyjaśniona powyżej, a ta praca dyplomowa mogła by być dobrym punktem wyjścia do kontynuacji badań na przedstawionym zagadnieniem. Do uzyskania wiarygodnych i bardziej uży-

tecznych wyników niewątpliwie przyczyniłoby się użycie wspomnianego już sprzętowego generatora ruchu. Duże ośrodki badawcze zajmujące się obszarem sieci komputerowych z pewnością dysponują odpowiednim i dokładnym sprzętem pomiarowym, który jest niezbędny w tego typu projektach. Również stworzenie większego środowiska testowego nie powinno być w ich przypadku problemem. Reasumując, przedstawione w mojej pracy: konfiguracja, testy parametrów sieci, możliwości protokołów OpenFlow i MPLS, a po części i otrzymane wyniki, opisujące szybkość działania tych technologii, mogą być dobrym punktem startowym dla badań obejmujących te zagadnienia tyle, że prowadzonych na większą skalę.

Dodatek A

Pliki konfiguracyjne

Listing A.1: R2 MPLS for Linux

```
#!/bin/bash

ifconfig eth4 down
ifconfig eth5 down
ifconfig eth6 down

ifconfig eth4 10.0.2.2 netmask 255.255.255.0 up
ifconfig eth6 172.16.10.2 netmask 255.255.255.0 up
ifconfig eth5 10.0.4.2 netmask 255.255.255.0 up

echo "1" >/proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter

mpls labelspace set dev eth4 labelspace 0
mpls ilm add label gen 2001 labelspace 0

var=‘mpls nhlfe add key 0 instructions push gen 1000 nexthop
   eth4 ipv4 10.0.2.3| grep key |cut -c 17-26‘
ip route add 172.16.20.0/24 via 10.0.2.3 mpls $var
```

Listing A.2: R3 MPLS for Linux

```
#!/bin/bash

ifconfig eth0 down
ifconfig eth1 down
ifconfig eth2 down

ifconfig eth0 10.0.6.3 netmask 255.255.255.0 up
ifconfig eth1 10.0.2.3 netmask 255.255.255.0 up

echo "1" >/proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter

mpls labelspace set dev eth1 labelspace 0
mpls ilm add label gen 1000 labelspace 0
var=`mpls nhlfe add key 0 instructions push gen 1001 nexthop
   eth0 ipv4 10.0.6.4 |grep key | cut -c 17-26`
mpls xc add ilm_label gen 1000 ilm_labelspace 0 nhlfe_key
   $var

mpls labelspace set dev eth0 labelspace 0
mpls ilm add label gen 2000 labelspace 0
var=`mpls nhlfe add key 0 instructions push gen 2001 nexthop
   eth1 ipv4 10.0.2.2 | grep key |cut -c 17-26`
mpls xc add ilm_label gen 2000 ilm_labelspace 0 nhlfe_key
   $var
```

Listing A.3: R3 Cicso MPLS

```
enable

conf t
mpls label range 10001 1048575 static 16 10000
end

conf t
mpls ip
interface FastEthernet0/0
ip address 10.0.6.3 255.255.255.0
mpls ip
no shut
end

conf t
interface FastEthernet0/1
ip address 10.0.2.3 255.255.255.0
mpls ip
no shut
end

conf t
mpls static crossconnect 1000 FastEthernet0/0 10.0.6.4 1001
mpls static crossconnect 2000 FastEthernet0/1 10.0.2.2 2001
end
```

Listing A.4: R4 Cisco MPLS

```
enable
conf t
mpls label range 10001 1048575 static 16 10000
end

conf t
mpls ip
interface FastEthernet0/0
ip address 172.16.20.4 255.255.255.0
mpls ip
no shut
end

conf t
interface FastEthernet0/1
ip address 10.0.6.4 255.255.255.0
mpls ip
no shut
end

conf t
ip route 172.16.10.0 255.255.255.0 10.0.6.3
mpls static binding ipv4 172.16.10.0 255.255.255.0 output
   10.0.6.3 2000
mpls static crossconnect 1001 FastEthernet0/0 172.16.20.20
   implicit-null
end
```

Listing A.5: S2 OpenFlow

```
#!/bin/bash

ifconfig eth2 172.16.20.2 netmask 255.255.255.0 up

./openflow/udatapath/ofdatapath ptcp:6634 -D -i eth2,eth4

dpctl add-flow tcp:127.0.0.1:6634 in_port=1,idle_timeout=0,
   actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 in_port=2,idle_timeout=0,
   actions=output:1
```

Listing A.6: S3 OpenFlow

```
#!/bin/bash

./openflow/udatapath/ofdatapath ptcp:6634 -D -i eth1,eth3

dpctl add-flow tcp:127.0.0.1:6634 in_port=1,idle_timeout=0,
   actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 in_port=2,idle_timeout=0,
   actions=output:1
```

Listing A.7: S4 OpenFlow

```
#!/bin/bash

ifconfig eth1 172.16.20.4 netmask 255.255.255.0 up

./openflow/udatapath/ofdatapath ptcp:6634 -D -i eth1,eth2

dpctl add-flow tcp:127.0.0.1:6634 in_port=1,idle_timeout=0,
   actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 in_port=2,idle_timeout=0,
   actions=output:1
```

Listing A.8: S2 Traffig Engineering (OF)

```
#!/bin/bash

ifconfig eth2 172.16.20.2 netmask 255.255.255.0 up

./openflow/udatapath/ofdatapath ptcp:6634 -D -i eth1,eth2,
   eth4

dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.10,
   nw_dst=172.16.20.20,idle_timeout=0,actions=output:3
dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.10,
   nw_dst=172.16.20.3,idle_timeout=0,actions=output:1
dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.20,
   nw_dst=172.16.20.10,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.3,
   nw_dst=172.16.20.10,idle_timeout=0,actions=output:2


dpctl add-flow tcp:127.0.0.1:6634 arp,in_port=3,idle_timeout
   =0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 arp,in_port=1,idle_timeout
   =0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 arp,in_port=2,idle_timeout
   =0,actions=output:1,output:3
```

Listing A.9: S3 Traffic Engineering (OF)

```
#!/bin/bash

ifconfig eth1 172.16.20.3 netmask 255.255.255.0 up

./openflow/udatapath/ofdatapath ptcp:6634 -D -i eth1,eth3

dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.20,
   nw_dst=172.16.20.10,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.10,
   nw_dst=172.16.20.20,idle_timeout=0,actions=output:1

dpctl add-flow tcp:127.0.0.1:6634 arp,in_port=1,idle_timeout
   =0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 arp,in_port=2,idle_timeout
   =0,actions=output:1
```

Listing A.10: S4 Traffic Engineering (OF)

```
#!/bin/bash

ifconfig eth1 172.16.20.4 netmask 255.255.255.0 up
ifconfig eth2 172.16.20.5 netmask 255.255.255.0 up

./openflow/udatapath/ofdatapath ptcp:6634 -D -i eth1,eth2,
   eth3

dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.10,
   nw_dst=172.16.20.20,idle_timeout=0,actions=output:1
dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.3,
   nw_dst=172.16.20.10,idle_timeout=0,actions=output:3
dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.20,
   nw_dst=172.16.20.10,idle_timeout=0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 ip,nw_src=172.16.20.10,
   nw_dst=172.16.20.3,idle_timeout=0,actions=output:2

dpctl add-flow tcp:127.0.0.1:6634 arp,in_port=2,idle_timeout
   =0,actions=output:1,output:3
dpctl add-flow tcp:127.0.0.1:6634 arp,in_port=3,idle_timeout
   =0,actions=output:2
dpctl add-flow tcp:127.0.0.1:6634 arp,in_port=1,idle_timeout
   =0,actions=output:2
```

Listing A.11: R2 Traffic Engineering (Cisco MPLS)

```
enable
conf t
mpls label range 10001 1048575 static 16 10000
end

conf t
mpls ip
interface FastEthernet0/0
ip address 10.0.4.2 255.255.255.0 secondary
ip address 172.16.10.2 255.255.255.0
mpls ip
no shut
end

conf t
interface FastEthernet0/1
ip address 10.0.2.2 255.255.255.0
mpls ip
no shut
end

conf t
ip route 10.0.6.0 255.255.255.0 10.0.4.4
ip route 172.16.20.0 255.255.255.0 10.0.2.3
mpls static binding ipv4 10.0.6.0 255.255.255.0 output
   10.0.4.4 4000
mpls static binding ipv4 172.16.20.0 255.255.255.0 output
   10.0.2.3 1000
mpls static crossconnect 2001 FastEthernet0/0 172.16.10.10
   implicit-null
mpls static crossconnect 3001 FastEthernet0/0 172.16.10.10
   implicit-null
end
```

Listing A.12: R3 Traffic Engineering (Cisco MPLS)

```
enable
conf t
mpls label range 10001 1048575 static 16 10000
end

conf t
mpls ip
interface FastEthernet0/0
ip address 10.0.6.3 255.255.255.0
mpls ip
no shut
end

conf t
interface FastEthernet0/1
ip address 10.0.2.3 255.255.255.0
mpls ip
no shut
end

conf f
ip route 172.16.10.0 255.255.255.0 10.0.6.4
mpls static binding ipv4 172.16.10.0 255.255.255.0 output
   10.0.6.4 3000
mpls static crossconnect 1000 FastEthernet0/0 10.0.6.4 1001
mpls static crossconnect 2000 FastEthernet0/1 10.0.2.2 2001
end
```

Listing A.13: R4 Traffic Engineering (Cisco MPLS)

```
enable
conf t
mpls label range 10001 1048575 static 16 10000
end

conf t
mpls ip
interface FastEthernet0/0
ip address 10.0.4.4 255.255.255.0 secondary
ip address 172.16.20.4 255.255.255.0
mpls ip
no shut
end

conf t
interface FastEthernet0/1
ip address 10.0.6.4 255.255.255.0
mpls ip
no shut
end

conf t
ip route 172.16.10.0 255.255.255.0 10.0.6.3
mpls static binding ipv4 172.16.10.0 255.255.255.0 output
   10.0.6.3 2000
mpls static crossconnect 1001 FastEthernet0/0 172.16.20.20
   implicit-null
mpls static crossconnect 3000 FastEthernet0/0 10.0.4.2
   implicit-null
mpls static crossconnect 4000 FastEthernet0/1 10.0.6.3
   implicit-null
end
```

Listing A.14: R2 Traffic Engineering (MPLS for Linux)

```
#!/bin/bash

ifconfig eth4 down
ifconfig eth5 down
ifconfig eth6 down

ifconfig eth4 10.0.2.2 netmask 255.255.255.0 up
ifconfig eth6 172.16.10.2 netmask 255.255.255.0 up
ifconfig eth5 10.0.4.2 netmask 255.255.255.0 up

echo "1" >/proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter

echo Setting MPLS for R2

mpls labelspace set dev eth4 labelspace 0
mpls ilm add label gen 2001 labelspace 0

var=‘mpls nhlfe add key 0 instructions push gen 1000 nexthop
   eth4 ipv4 10.0.2.3| grep key |cut -c 17-26‘
ip route add 172.16.20.0/24 via 10.0.2.3 mpls $var

mpls labelspace set dev eth5 labelspace 0
mpls ilm add label gen 3001 labelspace 0

var2=‘mpls nhlfe add key 0 instructions push gen 4000 nexthop
    eth5 ipv4 10.0.4.4| grep key |cut -c 17-26‘
ip route add 10.0.6.0/24 via 10.0.4.4 mpls $var2
```

Listing A.15: R3 Traffic Engineering (MPLS for Linux)

```
#!/bin/bash

ifconfig eth0 down
ifconfig eth1 down
ifconfig eth2 down

ifconfig eth0 10.0.6.3 netmask 255.255.255.0 up
ifconfig eth1 10.0.2.3 netmask 255.255.255.0 up

echo "1" >/proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter

mpls labelspace set dev eth1 labelspace 0
mpls ilm add label gen 1000 labelspace 0
var=`mpls nhlfe add key 0 instructions push gen 1001 nexthop
   eth0 ipv4 10.0.6.4 |grep key | cut -c 17-26`
mpls xc add ilm_label gen 1000 ilm_labelspace 0 nhlfe_key
   $var

mpls labelspace set dev eth0 labelspace 0
mpls ilm add label gen 2000 labelspace 0
var=`mpls nhlfe add key 0 instructions push gen 2001 nexthop
   eth1 ipv4 10.0.2.2 | grep key |cut -c 17-26`
mpls xc add ilm_label gen 2000 ilm_labelspace 0 nhlfe_key
   $var

var2=`mpls nhlfe add key 0 instructions push gen 3000 nexthop
    eth0 ipv4 10.0.6.4 |grep key | cut -c 17-26`
ip route add 172.16.10.0/24 via 10.0.6.4 mpls $var2
```

Listing A.16: R4 Traffic Engineering (MPLS for Linux)

```
#!/bin/bash

ifconfig eth0 down
ifconfig eth1 down
ifconfig eth2 down

ifconfig eth2 10.0.6.4 netmask 255.255.255.0 up
ifconfig eth1 172.16.20.4 netmask 255.255.255.0 up
ifconfig eth0 10.0.4.4 netmask 255.255.255.0 up

echo "1" >/proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter

var=‘mpls nhlfe add key 0 instructions push gen 2000 nexthop
   eth2 ipv4 10.0.6.3 |grep key | cut -c 17-26‘
ip route add 172.16.10.0/24 via 10.0.6.3 mpls $var

mpls labelspace set dev eth2 labelspace 0
mpls ilm add label gen 1001 labelspace 0

mpls labelspace set dev eth2 labelspace 0
mpls ilm add label gen 3000 labelspace 0
var2=‘mpls nhlfe add key 0 instructions push gen 3001 nexthop
    eth0 ipv4 10.0.4.2 |grep key | cut -c 17-26‘
mpls xc add ilm_label gen 3000 ilm_labelspace 0 nhlfe_key
   $var2

mpls labelspace set dev eth0 labelspace 0
mpls ilm add label gen 4000 labelspace 0
```

Listing A.17: R2 Failover (Cisco MPLS)

```
#!/bin/bash

ifconfig eth4 down
ifconfig eth5 down
ifconfig eth6 down

ifconfig eth5 10.0.2.2 netmask 255.255.255.0 up
ifconfig eth6 172.16.10.2 netmask 255.255.255.0 up
ifconfig eth4 10.0.4.2 netmask 255.255.255.0 up

echo "1" >/proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter

ip route del 172.16.20.0/24 via 10.0.2.3
var_default=‘mpls nhlfe add key 0 instructions push gen 4000
   nexthop eth4 ipv4 10.0.4.4| grep key |cut -c 17-26‘
ip route add 172.16.20.0/24 via 10.0.4.4 mpls $var_default

var_backup=‘mpls nhlfe add key 0 instructions push gen 1000
   nexthop eth5 ipv4 10.0.2.3| grep key |cut -c 17-26‘

mpls labelspace set dev eth5 labelspace 0
mpls ilm add label gen 2001 labelspace 0

mpls labelspace set dev eth4 labelspace 0
mpls ilm add label gen 3000 labelspace 0

default_route=1
echo "using default route"
while true
do
  sleep 1
  state=‘ethtool eth4 | grep "Link detected" | grep yes‘
  if [ -z "$state" ]
  then
    if [ $default_route -eq 1 ]
    then
      echo "using backup route"
      default_route=0
      ip route del 172.16.20.0/24
      ip route add 172.16.20.0/24 via 10.0.2.3 mpls
   $var_backup
    fi
  else
    if [ $default_route -eq 0 ]
    then
      echo "using default route"
      default_route=1
```

```
      ip route del 172.16.20.0/24
      ip route add 172.16.20.0/24 via 10.0.4.4 mpls
   $var_default
    fi
  fi
done
```

Listing A.18: R4 Failover (MPLS for Linux)

```
enable
conf t
mpls label range 10001 1048575 static 16 10000
end

conf t
mpls ip
interface FastEthernet0/0
ip address 10.0.6.4 255.255.255.0 secondary
ip address 172.16.20.4 255.255.255.0
mpls ip
no shut
end

conf t
interface FastEthernet0/1
ip address 10.0.4.4 255.255.255.0

mpls ip
no shut
end

conf t
ip route 172.16.10.0 255.255.255.0 10.0.4.2
ip route 172.16.10.0 255.255.255.0 10.0.6.3 20
mpls static binding ipv4 172.16.10.0 255.255.255.0 output
   10.0.4.2 3000
mpls static binding ipv4 172.16.10.0 255.255.255.0 output
   10.0.6.3 2000
mpls static crossconnect 1001 FastEthernet0/0 172.16.20.20
   implicit-null
mpls static crossconnect 4000 FastEthernet0/0 172.16.20.20
   implicit-null
end
```

Bibliografia

[1] Parca zbiorowa (red. prowadzący: Tomasz Janoś), *Vademecum teleinformatyka III, Rozdział 17*, 2004

[2] E. Rosen, A. Viswanathan, R. Callon, *Multiprotocol Label Switching Architecture*, RFC 3031, 2001

[3] Gary A. Donahue, *Wojownik sieci*, Rozdział 24, 2011

[4] *sourceforge.net/projects/mpls-linux/*

[5] I.Maravic, A.Smiljanic, *MPLS Implementation for the Linux Kernel*, 2012

[6] *Cisco Active Network Abstraction 3.7 Reference Guide*, Rozdział 14, 2010

[7] *OpenFlow Switch Specification, Version 1.0.0*, 2009

[8] Józef Muszyński, *OpenFlow – Nowe możliwości zarządzania ruchem w sieci*, www.networld.pl

[9] Michał Kisiel, *Konfiguracja ścieżek MPLS w Linuxie*, 2012

[10] *www.cisco.com/en/US/products/ps5855/products_tech_note09186a00801fc986.shtml*

[11] *www.openflow.org/wk/index.php/Ubuntu_Install*

[12] Praca zbiorowa (kierownik pracy: Mariusz Gajewski), *Realizacja usług Voice over IP i Video over IP w sieciach operatorskich i korporacyjnych*, 2006