

AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa magisterska

Protokoły OpenFlow i OVSDb w sieciach programowalnych SDN.

OpenFlow and OVSDb in programmable Software Defined
Networks.

Autor: Marek Ryznar
Kierunek studiów: Informatyka
Opiekun pracy: dr Michał Turek

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. O prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej <<sądem koleżeńskim>>.”, oświadczam, że niniejszą pracę dyplomową wykona łem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję promotorowi za wsparcie merytoryczne.
Oraz narzeczonej za wsparcie mentalne.

Spis Treści:

1.	Wprowadzenie	6
1.1.	Cele pracy	6
1.2.	Geneza pracy	7
1.3.	Zawartość pracy.....	7
2.	Wstęp teoretyczny.....	9
2.1.	SDN	9
2.2.	OVS	11
2.3.	OpenFlow	13
2.4.	OVSDB.....	14
2.5.	MEF.....	15
3.	Kryteria porównania	17
3.1.	Tagowanie VLAN	17
3.2.	Przypisywanie pakietów do EVC	19
3.3.	Definicja QoS	19
4.	Analiza protokołów na podstawie dokumentacji	20
4.1.	Dokładniejszy opis protokołów	20
4.1.1.	OpenFlow	20
4.1.2.	OVSDB	26
4.2.	Zestawienie protokołów na podstawie dokumentacji.....	27
4.2.1.	Tagowanie VLAN	27
4.2.2.	Przypisywanie pakietów do EVC.....	29
4.2.3.	Definicja QoS	30
5.	Zestawienie protokołów w rzeczywistych przypadkach użycia	32
5.1.	Mininet.....	32
5.2.	Dostępne narzędzia do konfigurowania przełączników OVS	36
5.3.	OpenDayLight	37

5.3.1.	Opis	37
5.3.2.	Uruchomienie kontrolera i połączenie z siecią mininet	39
5.3.3.	Analiza OpenFlow i OVSDb w kontrolerze	42
5.3.4.	Podsumowanie kontrolera	52
5.4.	ONOS	52
5.4.1.	Opis	52
5.4.2.	Uruchomienie kontrolera i połączenie z siecią mininet	55
5.4.3.	Analiza OpenFlow i OVSDb w kontrolerze	58
5.4.4.	Podsumowanie kontrolera	61
6.	Porównanie wyników oraz wnioski	62
6.1.	Podsumowanie zestawienia protokołów na podstawie dokumentacji	62
6.2.	Podsumowanie zestawienia protokołów w rzeczywistych przypadkach użycia	63
6.2.1.	Kontroler OpenDayLight	63
6.2.2.	Kontroler ONOS	63
7.	Podsumowanie	65

1. Wprowadzenie

Współcześnie pojęcie „informatyka” jest tak dużym zagadnieniem, że trudno znaleźć jedną osobę, która mogłaby się nazwać specjalistą od wszystkich jej aspektów. W związku z tym dzieli się ona na dziedziny, skupiające się wokół konkretnych problemów. Dziedzina, którą chciałem przedstawić w tej pracy to sieci komputerowe. Jest to jedna z kwestii, która musiała być zrealizowana w procesie digitalizacji, który doprowadził nas do ery komputerów, w jakiej dzisiaj żyjemy.

Jednym z najważniejszych standardów zdefiniowanych w specjalizacji sieci komputerowych jest OSI/ISO RM (*ISO Open Systems Interconnection Reference Model*), czyli siedmiowarstwowy model odniesienia dla większości protokołów komunikacyjnych. Warstwy druga (*łącza danych*) i trzecia (*sieciowa*) są to miejsca, w których *znajdują się* przełączniki i routery. Zestawienie i utrzymanie sieci przy użyciu wspomnianych urządzeń nigdy nie było prostym zadaniem.

Z pomocą w wykonaniu tego celu wkroczyła koncepcja sieci programowalnych, czyli SDN (*Software Defined Network*). Odbiorca niniejszej pracy zostanie zapoznany z ich zastosowaniem oraz z głównymi protokołami komunikacyjnymi używanymi w ich działaniu.

1.1. Cele pracy

Celem pracy jest porównanie dwóch głównych protokołów komunikacyjnych, czyli OVSDB (*Open vSwitch Database Management Protocol*) i OpenFlow działających w programowalnych sieciach komputerowych. Kryterium porównania są elementy składowe usług zgodnych ze standardem Carrier Ethernet 2.0, definiowanych przez organizację MEF (*Metro Ethernet Forum*).

Na samym początku rozważań protokoły zostaną porównane na podstawie ich specyfikacji. Następnie postawiona zostanie wirtualna sieć komputerowa typu SDN pozwalająca na zestawienie możliwości protokołów w praktyce. Wyniki symulacji zostaną przedstawione za pomocą dedykowanej do tego aplikacji.

1.2. Geneza pracy

Pomysł na zrealizowanie porównania protokołów OpenFlow i OVSDb w pracy magisterskiej narodził się podczas mojego udziału w tworzeniu PoC (*Proof of Concept*) na konferencję organizacji MEF w Listopadzie 2016 roku [18]. Moja kontrybucja do tego projektu zaczęła się podczas pracy w firmie Amartus w Krakowie.

Efektem końcowym PoC miało być przedstawienie stworzenia usługi E-Line w środowisku SDN, gdzie urządzenia sieciowe pochodzą od wielu dostawców. W przygotowanym projekcie wykorzystany został orkiestrator sieciowy *Cloudify*, kontroler SDN *OpenDayLight* oraz dwa rodzaje przełączników – Cisco CRS1000V oraz wirtualne przełączniki OVS [18].

Moim zadaniem było stworzenie sieci urządzeń wirtualnych Open vSwitch (*OVS*) i napisanie modułu w projekcie UniMgr kontrolera OpenDayLight odpowiedzialnego za zarządzanie nimi. Dzięki pomocy kolegów z pracy udało mi się stworzyć wirtualną sieć za pomocą programu *mininet*. W tak stworzonej sieci musiałem wymyślić sposób stworzenia wirtualnego połączenia (*EVC*) pomiędzy interfejsami krańcowymi sieci mininet (*UNIs*), czyli portami przełączników OVS, a następnie zaimplementować wykonujący to sterownik w kontrolerze ODL.

Konfigurację przełączników OVS wykonać można właśnie za pomocą protokołów OpenFlow i OVSDb. Co prawda służą do konfiguracji innych rzeczy, lecz na wyższym poziomie abstrakcji, w niektórych przypadkach obydwa mają możliwość wykonania podobnych akcji. W tej pracy postanowiłem porównać ich działanie podczas konfiguracji atrybutów usług zdefiniowanych przez organizację MEF (np. sposób obsługi pakietów z tagiem VLAN).

1.3. Zawartość pracy

W rozdziale 2 znajduje się wstęp teoretyczny zapewniający wyjaśnienie terminologii używanej w niniejszej pracy. Następnie przedstawione zostają kryteria porównania protokołów OpenFlow i OVSDb. W rozdziale 4 opisane są dokładniej opisywane protokoły, po czym następuje ich analiza porównawcza na podstawie dokumentacji i innych źródeł. W kolejnym etapie pracy zaprezentowany jest efekt zestawienia protokołów w rzeczywistych przypadkach użycia, czyli proces tworzenia sieci wirtualnej oraz połączenie z nią wirtualnych kontrolerów (najpierw OpenDayLight a później ONOS).

W następnym rozdziale znajduje się zestawienie wszystkich wyników porównania i wysnute zostają wnioski. Ostatni rozdział zawiera podsumowanie całej pracy.

2. Wstęp teoretyczny

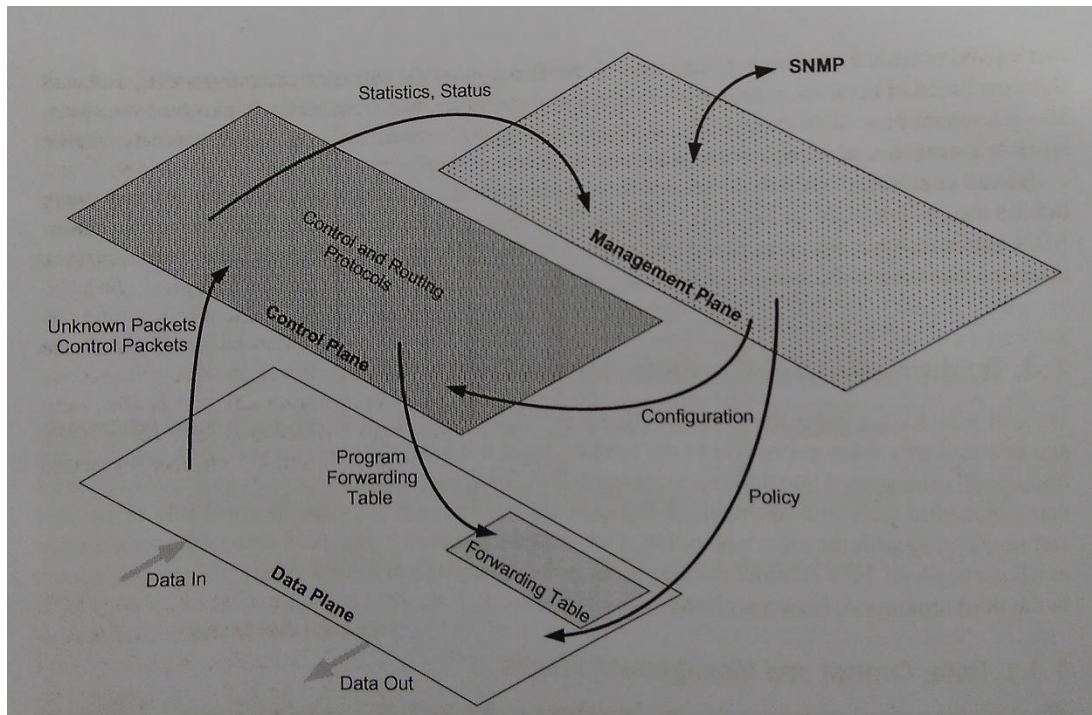
2.1. SDN

Urządzenia sieciowe są od wielu lat instalowane i zarządzane w licznych firmach dostarczających usługi sieciowe. Mostki, przełączniki i routery od zawsze były używane w wielu środowiskach wykonując funkcję filtrowania i przekazywania pakietów przez sieć komputerową. Pomimo wielu dobrze działających tradycyjnych technologii, wielkość i skomplikowanie współczesnych zestawień sieciowych buduje potrzebę innowacji w tej dziedzinie. Głównymi powodami są wiecznie rosnące koszty posiadania i zarządzania sprzętem sieciowym oraz wzrastająca potrzeba na nowoczesne centra danych. Obecnie wiele firm sieciowych powoli odchodzi od tradycyjnych metod w kierunku bardziej wolno dostępnego i otwartego na innowacje paradygmatu SDN. [1]

Aby zrozumieć fenomen sieci programowalnych trzeba najpierw zdefiniować tradycyjną architekturę przełącznika. Przedstawić można ją w świetny sposób za pomocą trzech warstw:

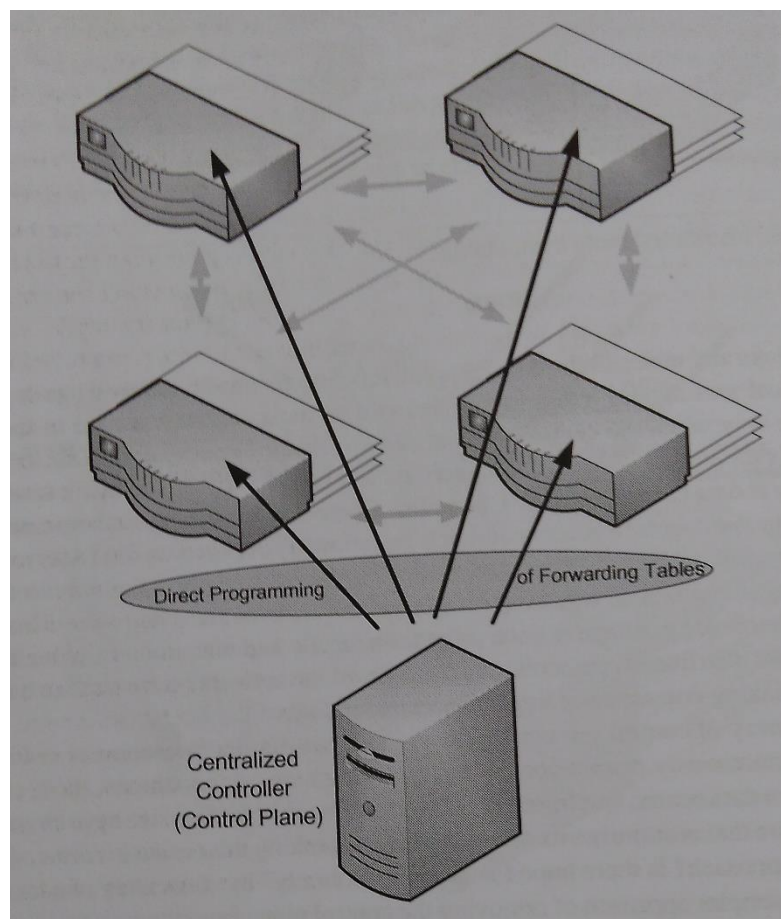
- a) **Data Plane** – składa się z portów odpowiedzialnych za pobieranie pakietów oraz z tablicy skojarzeń (*forwarding table*). Warstwa ta jest odpowiedzialna za modyfikację nagłówka ramki oraz buforowanie i przekazywanie pakietów,
- b) **Control Plane** – głównym zadaniem tej warstwy jest utrzymywanie aktualnych informacji w tablicy skojarzeń, tak aby data plane mogło kierować ruchem niezależnie bez ingerencji innych warstw. Tutaj właśnie procesowane są wszystkie protokoły odpowiedzialne za zmianę rekordów w tablicy skojarzeń, czyli odpowiedzialne za zarządzanie topologią sieci (np. RIP, OSPF, BGP itd.),
- c) **Management Plane** – jest to ostatnia (najwyższa) warstwa modelu, odpowiedzialna za zarządzanie urządzeniem bezpośrednio przez administratora (np. Poprzez protokół SNMP).

Taki model warstwowy został przedstawiony na rysunku poniżej.



Rysunek 1: Role data, control i management planes [1].

Warstwa control plane jest nieustannie bombardowana przez protokoły do wyznaczania ścieżek pakietów. Faktem jest, że w wielkich sieciach center danych, około 30% czasu ruterów jest wykorzystywane do śledzenia topologii sieciowej [1]. Mimo zaawansowanych protokołów trasowania, możliwe jest usprawnienie obecnych sieci, tym rozwiązaniem jest użycie sieci programowalnych. Polega to na tym, że warstwa control plane nie jest zarządzana przez protokoły trasowania, ale przez centralnie zlokalizowany inteligentny kontroler posiadający informacje o topologii sieciowej. Na rysunku poniżej przedstawiona jest wersja sieci zarządzana przez kontroler SDN.



Rysunek 2: Zcentralizowany kontroler zarządzający warstwą control plane [1].

Prostota tego rozwiązania wynika z tego, że topologia sieciowa, jest stabilna i pod ścisłą kontrolą administratora sieciowego w jednym zcentralizowanym miejscu. Dodatkowym atutem przemawiającym za tym rozwiązaniem jest szybka diagnoza potencjalnych błędów w dużej sieci komputerowej.

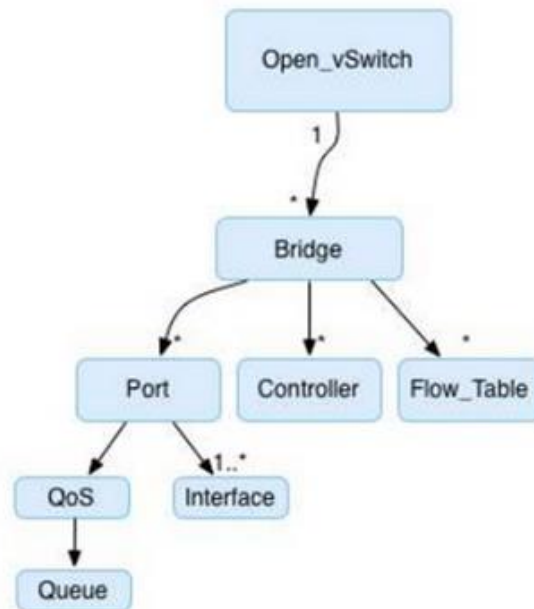
2.2.OVS

Kolejnym ważnym pojęciem, które będzie się często przewijało w niniejszej pracy jest OVS (*Open Virtual Switch*), czyli wirtualny przełącznik. Jest to oprogramowanie przedstawiające wielowarstwowy switch, udostępniany na podstawie darmowej licencji (*Open Source Apache 2 License*). Jest doskonale dostosowany, aby pełnić funkcję wirtualnego przełącznika w środowisku maszyn wirtualnych. Jest stworzony do automatyzacji sieci, przy czym wspiera obecne standardy i protokoły (np. NetFlow, Sflow, IPFIX, RSPAN, LACP, 802.1ag). OVS wspiera wiele bazujących na linuxie technologii takich jak Xen/XenServer, KVM czy VirtualBox [2].

OVS jest przeznaczony do środowisk z wieloma serwerami. Stan urządzeń wirtualnych w jednostce sieciowej (np. w maszynie wirtualnej) powinien być prosty

do zreprodukowania w innym miejscu. Wirtualne przełączniki to umożliwiają, poprzez możliwość migrowania tablic skojarzeń, listy kontroli dostępu (ACLs), polityki QoS itd [2].

Struktura danych wirtualnego przełącznika OVS ukazana jest na rysunku poniżej.



Rysunek 3: Struktura danych OVS'a [2].

Na powyższej ilustracji możemy zauważyć, że wirtualny przełącznik posiada trzy podstawowe elementy:

- a) Port – interfejs odpowiedzialny za połączenie OVSa z innymi urządzeniami. Jego konfiguracja jest ustawiana przez protokół OVSDb. W czasie działania wirtualnego przełącznika można w dowolnym czasie dodawać nowe porty i subinterfejsy,
- b) Controller – każdy wirtualny switch może być zarządzany przez jeden lub więcej kontrolerów SDN,
- c) Flow_Table – tablica reguł OpenFlow, czyli odpowiednik tablicy skojarzeń w tradycyjnym przełączniku (z tym, że posiada większe możliwości).

Najnowsza wersja oprogramowania OVS to 2.7.0 i ta wersja będzie wykorzystywana w zestawieniu sieci wirtualnej.

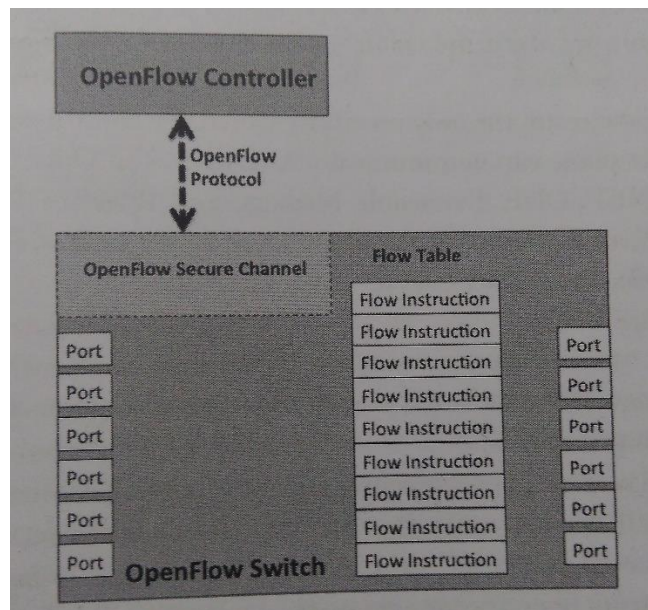
2.3. OpenFlow

Wraz z powstaniem sieci programowalnych, na scenę wkroczył protokół komunikacyjny OpenFlow. Został on zdefiniowany w ramach standardów wyspecyfikowanych przez organizację ONF (*Open Networking Foundation*), założoną na początku roku 2011 przez takie firmy jak Microsoft, Google, Verizon, Yahoo!, Deutsche Telecom i Facebook [4]. Protokół ten ma za zadanie komunikować warstwę danych (*data plane*) w przełączniku z warstwą *control plane* w kontrolerze. Aby kontroler mógł się porozumieć z switch'em za pomocą opisywanego protokołu, ten musi posiadać do niego wsparcie. Na teraźniejszym rynku sieci komputerowych istnieje coraz więcej przełączników wspierających protokół OpenFlow. Najbardziej znane są przełączniki wirtualne takie jak opisywany wcześniej OVS lub mniej znany *Indigo* stworzony przez firmę *BigSwitch* [1]. Niektóre firmy produkujące sprzęt sieciowy również wyposażyły swoje przełączniki o wsparcie dla protokołu OpenFlow (tzw. switch'e hybrydowe). Warto tutaj wspomnieć, że kontrolery SDN komunikują się z urządzeniami zarządzanymi nie tylko za pomocą protokołu OpenFlow, używają również innych protokołów by być w stanie porozumiewać się z jak największą liczbą urządzeń (SNMP, Netconf itd.).

OpenFlow powstał w celu ujednolicenia sposobu komunikacji kontrolera z przełącznikami. Na wysokim poziomie abstrakcji można powiedzieć, że standard OpenFlow definiuje następujące komponenty potrzebne do komunikacji kontroler - przełącznik:

- a) Kontroler,
- b) Interfejs OpenFlow w przełączniku,
- c) Kanał do bezpiecznej komunikacji,
- d) Tabele przepływów (*flow table*) zawierającą instrukcje przepływów[4].

Powyżej wymienione komponenty wizualizuje rysunek 4:



Rysunek 4: Podstawowe komponenty OpenFlow [4].

OpenFlow sam w sobie nie potrafi manipulować właściwościami przełączników i ich portów, odpowiedzialny jest tylko za modyfikowanie rekordów w tablicach skojarzeń OpenFlow (flow tables). W tym miejscu warto więc wspomnieć, że ONF stworzył protokół, który to umożliwia, nazywa się OF-CONFIG, jednakże w tej pracy nie będzie on opisywany, gdyż w porównaniu do OVSDB ma dużo mniejsze możliwości.

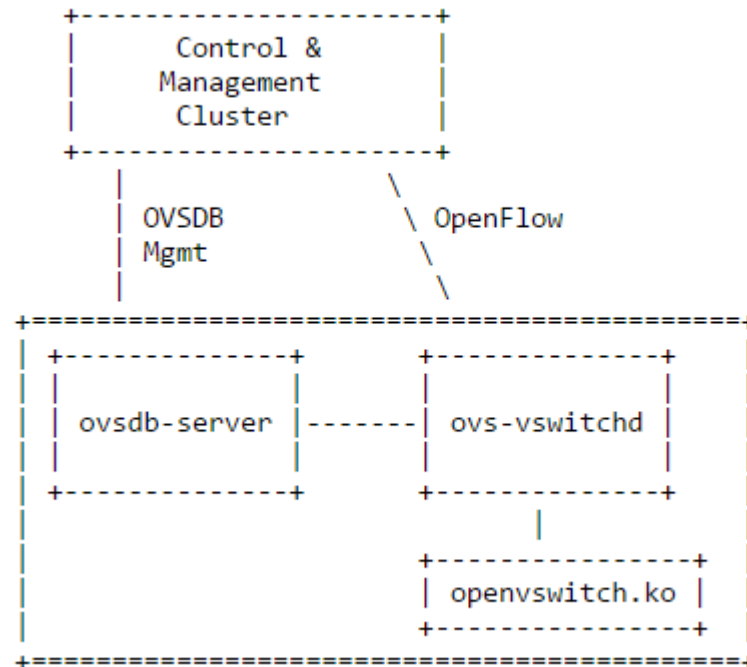
Więcej właściwości OpenFlow zostanie omówione w dalszych rozdziałach.

2.4. OVSDB

Jak już to wcześniej było wspominane, kontroler SDN może komunikować się z przełącznikami za pomocą wielu protokołów komunikacyjnych, jednym z nich jest OVSDB (*Open vSwitch Database Management Protocol*) stworzony przez firmę Nicra (później przejętą przez VMware). Został on stworzony wraz z stworzeniem wirtualnych przełączników OVS, jako protokół ich zarządzania.

Istnieje ogólne przekonanie, że kontroler może porozumiewać się z wirtualnym przełącznikiem tylko za pomocą protokołu OpenFlow, ale jeżeli chodzi o konfigurację, tu wkracza OVSDB. Za jego pomocą można dodawać, modyfikować i usuwać wirtualne przełączniki OVS, ich porty i interfejsy. Możliwe jest zarządzanie urządzeniem OVS, wyłącznie przy użyciu protokołu OVSDB [5].

Na poniższym rysunku widać interfejsy wirtualnego przełącznika OVS, gdzie *ovsdb-server* to serwer bazy danych protokołu OVSDb, *ovs-vswitchd* to daemon vswitch'a i *openvswitch.ko*, czyli moduł kernela wykonujący szybkie przekierowywanie ścieżek.



Rysunek 5: Interfejsy przełącznika OVS [6]

Wraz z pojawieniem się na rynku OVSDb, niektóre firmy postanowiły wprowadzić wsparcie dla tego protokołu. Do niniejszych firm należą min. *Cumulus*, *Arista* i *Dell* [5]. Więcej o OVSDb zostanie powiedziane w późniejszych rozdziałach.

2.5.MEF

Aby dopełnić wstęp teoretyczny do niniejszej pracy, trzeba powiedzieć, czym jest MEF. Jest to skrót od *Metro Ethernet Forum* i jest to międzynarodowa organizacja zrzeszająca ponad 220 przedsiębiorstw zajmujących dostarczaniem usług sieciowych, produkcją sprzętu sieciowego itd. Zajmuje się ona przyspieszeniem zastosowania sieci komputerowych i usług typu *Carrier-class Ethernet* [7].

Organizacja MEF definiuje w jaki sposób dostawcy usług sieciowych powinni tworzyć usługi. Zanim wymienione zostaną typy usług, trzeba wyjaśniona musi być terminologia używana przy ich definicji. Najważniejsze pojęcia to:

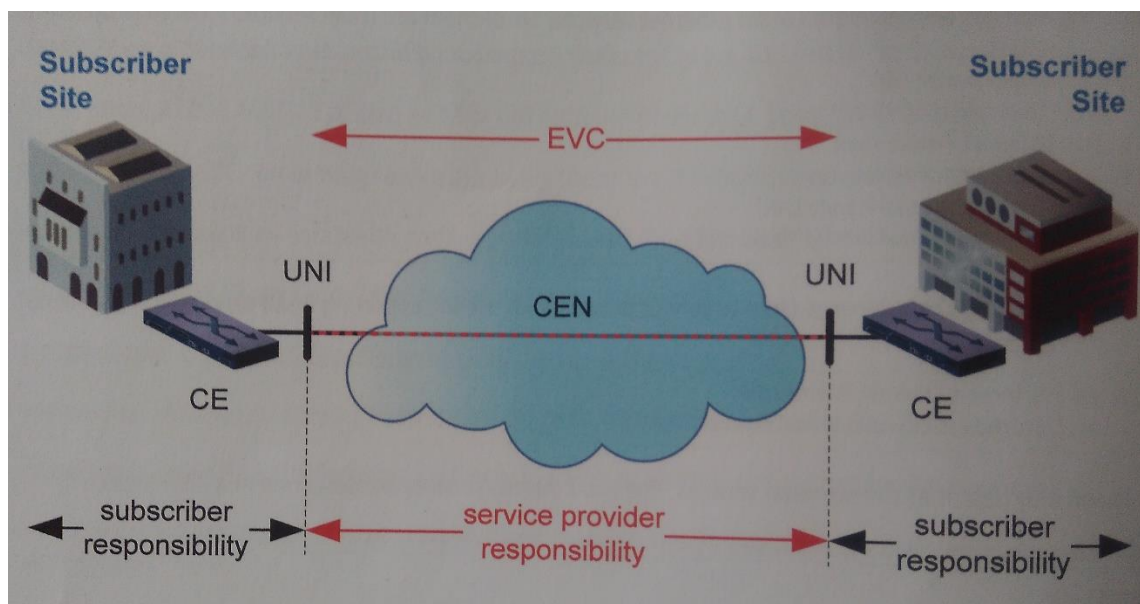
- a) CEN (*Carrier Ethernet Network*) – sieć dostawcy usług,

- b) CE (*Customer Edge equipment*) – sprzęt po stronie subskrybenta usługi, podłączona do CEN,
- c) UNI (*User Network Interface*) - to punkt fizycznego rozgraniczenia pomiędzy domenami klienta i dostawcy usługi,
- d) EVC (*Ethernet Virtual Connection*) – logiczna reprezentacja połączenia w usłudze pomiędzy dwoma lub więcej punktami końcowymi UNI.

Znając terminologię opisującą usługi, możemy wymienić trzy podstawowe typy usług definiowane przez standard MEF:

- a) E-Line – usługa definiująca bezpośrednie połączenie między dwoma punktami końcowymi UNI,
- b) E-LAN – usługa umożliwiająca na więcej niż dwa UNI, gdzie każdy z nich może przekazywać ramki ethernetowe do każdego innego UNI,
- c) E-Tree – usługa łącząca wiele UNI, posiada strukturę drzewiastą, przy czym zapobiega przekazywaniu ramek ethernetowych w kierunku od interfejsów użytkownika o roli Liść (*Leaf*) do tych o roli Korzeń (*Root*). Przekazywanie ramek w odwrotnym kierunku jest możliwe.

Na rysunku poniżej znajduje się przykładowa usługa E-Line.



Rysunek 6: Podstawowa usługa MEF [7].

MEF do wszystkich wymienionych wyżej usług definiuje atrybuty, takie jak jakość usługi (*Quality of Service*), ramki usług na podstawie ramek Ethernet itd. Właśnie tego typu

atrybuty będą wybrane jako kryterium porównania protokołów OpenFlow i OVSDb w niniejszej pracy i zostaną one dokładniej opisane w następnym rozdziale.

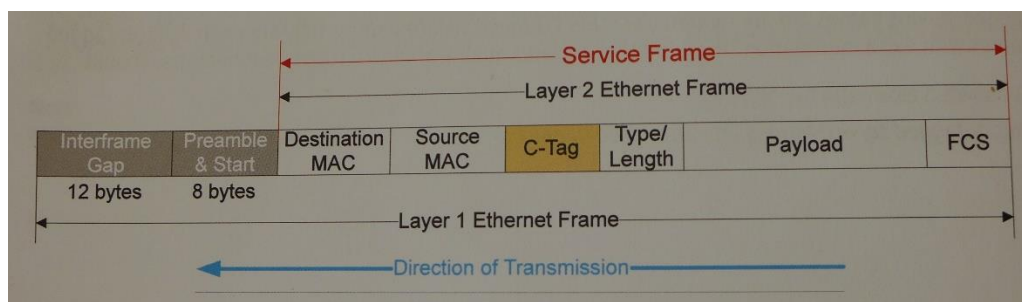
3. Kryteria porównania

Przy każdym porównaniu dwóch lub więcej obiektów potrzebne są kryteria względem, których nastąpi zestawienie. W tej pracy kryteriami porównania są atrybuty usług zdefiniowanych przez standard CarrierEthernet 2.0 MEF. Wybrane atrybuty są zwięźle opisane w poniższych podpunktach.

3.1. Tagowanie VLAN

Usługi MEF definiowane są jako niezależne od obecnych technologii sieciowych pozwalając dostawcom usług na ich stworzenie przy użyciu dowolnych, odpowiadających im technologii. Jednakże definicja usług sieciowych według MEF wymaga, aby połączenie pomiędzy klientem (*CE*) a siecią dostawcy usługi (*CEN*) była zgodna ze standardem IEEE 802.3 [7].

MEF definiuje ramkę danych, jako ramkę ethernetową warstwy drugiej modelu OSI/ISO. Tak zdefiniowana ramka nazywa się ramką usługi (*service frame*) i posiada wszystkie pola od docelowego adresu MAC (*Destination MAC Address*) do pola FCS (*Frame Check Sequence*), lecz nie zawiera komponentów ramki z warstwy pierwszej. Opisana ramka znajduje się na rysunku numer 7.



Rysunek 7: Ramka usługi [7]

Według standardu MEF ramki można rozróżniać między innymi na podstawie pola C-TAG (*Customer TAG*). Pole to zajmuje 4 bajty i składa się z następujących pól:

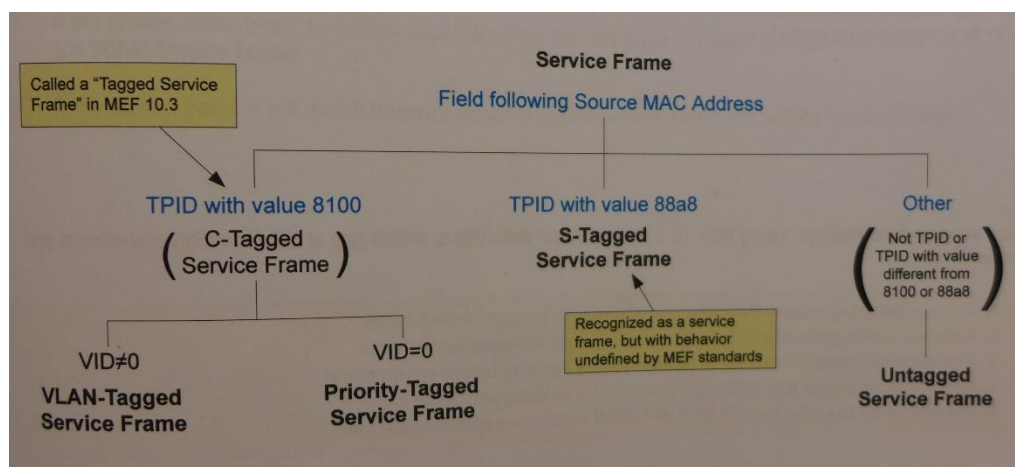
- TPID (*Tag protocol identifier*) – 16 bitowe pole definiujące typ tagu,
- TCI (*Tag control information*) – pole składające się z 3 składowych:

- a. PCP (*Priority code point*) – 3 bitowe reprezentujące priorytet na skali od 0 do 7,
- b. DEI (*Drop Eligibility Indicator*) – 1 bit mówiący czy ramka nadaje się do odrzucenia czy nie,
- c. VID (VLAN Identifier) – 12 bitów reprezentujących 4094 wartości identyfikujący, do jakiej sieci VLAN ramka należy.

MEF 10.3 definiuje trzy formaty ramek:

- a) Ramka nieotagowana – nie posiadająca żadnego tagu ethernetowego,
- b) Ramka tagowana przez VLAN – ramka C-TAG, gdzie TPID=8100, a VID jest różny od 0,
- c) Ramki priorytetowe - ramka C-TAG, gdzie TPID=8100, a VID jest równy 0.

Oprócz ramek tagowanych przez klienta (C-Tagged) MEF rozpoznaje również dodatkowe pole S-TAG, czyli tag VLAN należący do usługi. Pole to składa się z takich samych komponentów, co pole C-TAG. Tak tagowane ramki są rozpoznawane przez MEF jako ramki usługi, ale niestety jak narazie nie mają zdefiniowanego zachowania. Poniższy rysunek w dobry sposób ukazuje rozpoznawanie taga VLAN ze względu na pole TPID [7].



Rysunek 8: Rodzaj tagu w zależności od TPID [7].

Podsumowując, ramka może posiadać jedno lub więcej pole VLAN definiujące do jakiej podsieci wirtualnej przynależy ramka. Tag ten może być definiowany przez klienta (C-TAG) lub przez dostawcę usługi (S-TAG). Porównując protokoły niniejszym kryterium sprawdzana będzie *umiejętność* protokołów do nadawania tagu VLAN ramce, filtrowania ramek na jego podstawie, edytowania i usuwania. Ponadto sprawdzone zostanie, czy za pomocą wybranego protokołu można nadać więcej niż jeden tag VLAN, czyli czy wspierają

mechanizm IEEE 802.1 Q-in-Q zazwyczaj używany do nadania tagu VLAN przez dostawcę usługi (S-TAG).

3.2. Przypisywanie pakietów do EVC

Ramki usługi są przypisywane do EVC na dwa podstawowe sposoby. Pierwszy z nich jest bazowany na portach (*port-based*), co znaczy że wszystkie ramki usługi przy każdym z UNI są przypisywane do jednego EVC. Drugi sposób polega na wybieraniu wyjściowego EVC na podstawie tagu VLAN (*VLAN-based*) [7]. W tym kryterium protokoły będą porównywane na podstawie ich możliwości do przypisywania pakietów do EVC na podstawie portów i na używając tagów VLAN.

3.3. Definicja QoS

QoS jest skrótem od *Quality of Service* czyli jakość usługi. Według standardu MEF termin ten odnosi się do wydajności dostarczania ramek usługi, co zostało zdefiniowane w SLS (*Service Level Specification*) używając atrybutów EVC dotyczących wydajności usługi. Te atrybuty to dostępność, elastyczność, współczynnik strat ramki i opóźnienia ramek [7].

Aby uzyskać dany QoS, trzeba skorzystać z zarządzania ruchem (*traffic management*), czyli zbioru mechanizmów, narzędzi i polityk używanych przez dostawcę usługi. Jednym z mechanizmów używanych do uzyskania QoS są profile przepustowości (*Bandwidth profiles*). Używane są one do ograniczenia obciążenia usługi *Carrier Ethernet*.

W tym kryterium protokoły będą głównie porównywane pod względem stopnia implementacji profili przepustowości. Jeżeli któryś z protokołów będzie posiadał możliwość ustawienia jakiegoś parametru QoS, zostanie to porównane do możliwości drugiego protokołu w tym zakresie.

4. Analiza protokołów na podstawie dokumentacji

W niniejszym rozdziale znajduje się porównanie protokołów OpenFlow oraz OVSDb, gdzie najpierw zostanie ono wykonane według specyfikacji. Zestawienie ich będzie wykonane według kryteriów zdefiniowanych w poprzednim rozdziale.

4.1. Dokładniejszy opis protokołów

Aby być w stanie porównać działanie protokołów według danych kryteriów, trzeba przedstawić to, w jaki sposób protokoły wykonują podstawowe funkcje. W tym podrozdziale OpenFlow i OVSDb zostaną opisane w nieco szerszym zakresie niż było to zrobione we wstępie teoretycznym.

4.1.1. OpenFlow

OpenFlow został stworzony przez organizację ONF, która wydaje wiele dokumentów specyfikujących ten protokół. Niniejszy opis protokołu jest bazowany na podstawie dokumentu *OpenFlow Switch Specification*, który opisuje OpenFlow, jako protokół służący komunikacji kontrolera SDN z przełącznikiem. Dodatkowo określa wymagania, co do switch'a, z którym się komunikuje. ONF wydał już kilka wersji tej specyfikacji, w tej pracy wykorzystana zostanie najnowsza 1.5.0 (0x06) wydana 19 Grudnia 2014 roku [8].

Jak było to wspomniane we wstępie teoretycznym, podstawowe komponenty OpenFlow to switch i kontroler. O ile kontroler jest istotny w architekturze SDN, tak w architekturze OpenFlow jest drugoplanowy, ponieważ protokół nie ma nic do powiedzenia w kwestii decyzji, jakie instrukcje są przesyłane do przełącznika, jest to rozstrzygane na poziomie kontrolera lub na wyższych warstwach.

Przełącznik jest miejscem, w którym dzieje się większość akcji z punktu widzenia protokołu. Przetwarzanie pakietów odbywa się przez tabele: *flow tables*, *group table* i *meter table*. Rekordy w tabelach inicjalizowane są poprzez instrukcje podawane przez kontroler poprzez kanał OpenFlow. Pakiety wchodzi i wychodzą przez porty, tak jak w tradycyjnym przełączniku [4].

OpenFlow definiuje 3 podstawowe typy portów:

- a) Porty fizyczne – odwołują się bezpośrednio do interfejsów sprzętowych,

- b) Porty logiczne – w tym terminie zawierają się wszystkie logiczne interfejsy napotykane w tradycyjnych przełącznikach, np. interfejs loopback, interfejs null, MPLS LSP itd. Z perspektywy OpenFlow port logiczny jest traktowany tak samo jak fizyczny,
- c) Porty zarezerwowane – używane są do wewnętrznego przetwarzania pakietów. Porty te mogą być wymagane lub opcjonalne.

Porty wymagane:

- ALL - oznacza wszystkie porty oprócz wejściowego,
- CONTROLLER – port do bezpiecznego kanału łączącego przełącznik z kontrolerem,
- TABLE – wejściowy port do tzw. *pipeline*, czyli miejsca, w którym znajdują się tabele,
- IN_PORT – używany jest, kiedy pakiet musi być przesłany do portu, którym przyszedł,
- ANY – może reprezentować jakikolwiek port lub wszystkie.

Porty opcjonalne:

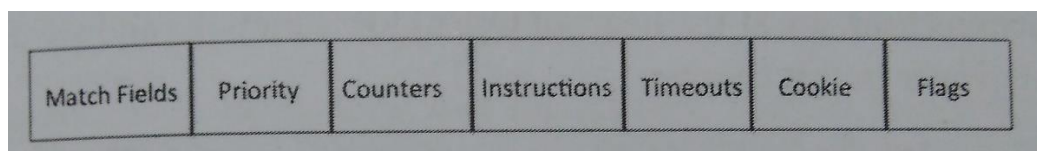
- LOCAL – wejściowy i wyjściowy port należący do lokalnego stosu przełącznika,
- NORMAL – wspierany tylko w switch'ach hybrydowych i jest to wyjściowy port przekierowujący pakiety do normalnego przetwarzania przełącznika,
- FLOOD – tak jak NORMAL, wspierany tylko przez switch'e hybrydowe. Przekazuje pakiet na wszystkie interfejsy fizyczne lub na pewną ich podgrupę[4].

Przełącznik OpenFlow komunikuje się z kontrolerem poprzez tzw. *Secure Channel*, który korzysta z TCP na porcie 6653, co zostało zdefiniowane przez IANA (*Internet Assigned Numbers Authority*). Switch inicjalizuje połączenie z kontrolerem podczas startu lub restartu, które zazwyczaj jest szyfrowane poprzez protokół TLS (*Transport Layer Security*) [4].

Jak już było to wspomniane, przetwarzanie pakietów odbywa się poprzez 3 tabele, pierwszą z nich jest *flow table*. Składa się ona z następujących pól:

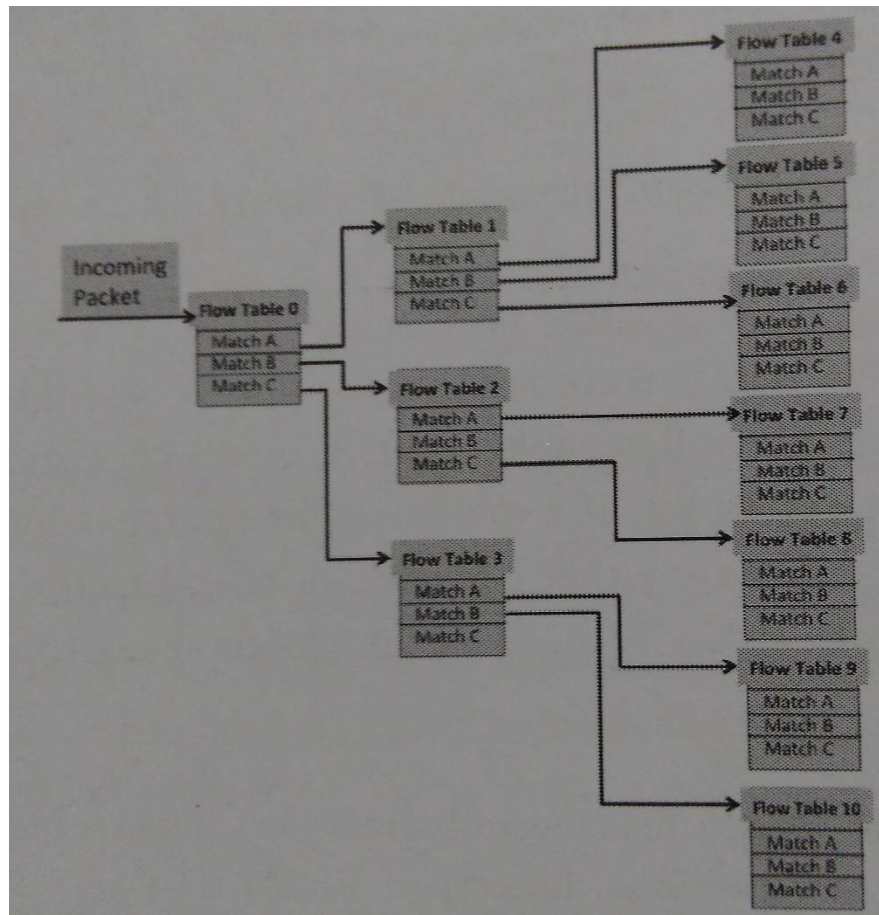
- a) *Match fields* – określa, pod jakimi warunkami pakiet jest dopasowywany. Może to być kombinacja portów wchodzących, nagłówek Ethernet lub IPv4, metadane itd.,
- b) *Priority* – definiuje kolejność rekordów,
- c) *Counter* – zapisuje statystyki dopasowanych pakietów,
- d) *Instruction* – specyfikuje, jakie akcje mają być wykonane po dopasowaniu pakietu,
- e) *Timeouts* – specyfikuje maksymalny czas istnienia rekordu (jeśli nie jest używany),
- f) *Cookie* – używany przez kontroler do filtrowania rekordów,
- g) *Flags* – służy do zmieniania sposobu dodawania rekordów [4].

Format rekordu tabeli przepływu zilustrowany jest na rysunku poniżej:



Rysunek 9: Format rekordu tabeli przepływu [4].

W pierwszej wersji protokołu (1.0.0) przełącznik OpenFlow mógł posiadać tylko jedną tabelę przepływu. Od wersji 1.1.0 switch może posiadać więcej takich tabel. Wraz z tym został wprowadzony mechanizm o nazwie *pipeline processing* odpowiadający za tworzenie hierarchii przetwarzania tabel przy użyciu prostych warunków „if-then-go”. Wszystkie przychodzące pakiety muszą być przetworzone przez tabele od id równym 0, po czym mogą być przekazane do kolejnych tabel. W dobry sposób przedstawia to rysunek 10.



Rysunek 10: Przetwarzanie pakietu przez wiele tabel [4].

Oprócz tabel przepływu OpenFlow definiuje tzw. *group tables*, w których rekordy reprezentują grupę pakietów, które powinny być traktowane w ten sam sposób. Tworząc pakiet, jako członek pewnej wybranej, zdefiniowanej wcześniej grupy, akcje mogą być wykonywane w bardziej wydajny sposób w porównaniu do tradycyjnego sposobu przepływu [4].

Ostatnią z opisywanych tabel jest wprowadzona w wersji 1.3.0 *meter table*, odpowiedzialna za tworzenie prostych mechanizmów QoS, które mierzą szybkość przepływu i na jego podstawie nakładają pewien limit. Tabela ta zostanie lepiej omówiona w dalszych podpunktach [4].

Wszystkie wiadomości OpenFlow są enkapsulowane wraz z nagłówkiem. Pola w nim zawarte to:

- a) *Version* – wersja według standardu *OpenFlow Switch Specification*,
- b) *Type* – typ wiadomości, dzieli się na trzy kategorie: *Controller-to-Switch* (np. FEATURE REQUEST, PACKET OUT, FLOW MOD, TABLE MOD, GROUP MOD, METER MOD), *Asynchronous* (np. PACKED IN, ERROR) i *Symmetric* (np. HELLO, ECHO REQUEST, ECHO REPLY),

- c) *Length* – długość wiadomości,
- d) *Transaction ID (XID)* – id wiadomości[4].

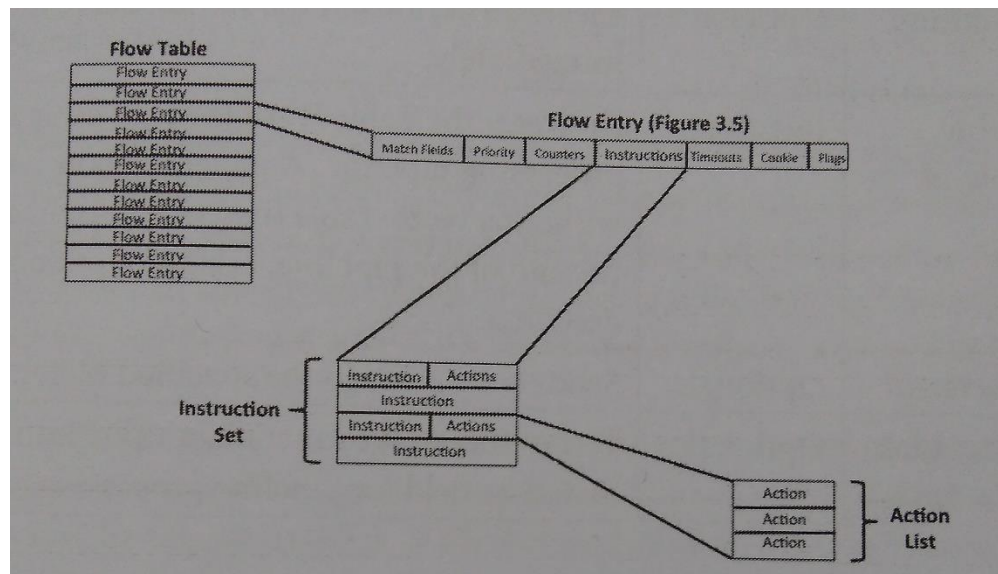
Istnieje kilka struktur używanych w wiadomości OpenFlow opisujących elementy przełącznika:

- a) *Port Structures* – definiuje porty,
- b) *Queue Structures* – każda kolejka składa się z:
 - 32 bitowego numeru portu,
 - 32 bitowego id kolejki,
 - Minimalna i maksymalna szybkość przepływu na porcie,
 - Struktura dopasowywania przepływu (*Flow Match*), która opisuje dopasowywanie pakietów lub rekordów przepływu (*flow entries*) [4].

Po opisanu najważniejszych komponentów, można przystąpić do objaśnienia tego, w jaki sposób przepływy są przetwarzane w przełączniku OpenFlow. Warunki dopasowania w rekordzie tabeli przepływu dzielą się na trzy kategorie:

- a) *Flow Match* – przepływ jest identyfikowany, jako kombinacja parametrów takich jak port wchodzący, adres źródłowy i docelowy, bity CoS (*Class of Service*) i porty wyższych warstw. Bazując na tych parametrach przepływ może być dopasowywany,
- b) *Header Match* – przepływ może być również dopasowywany na podstawie zawartości nagłówek pakietów (warstwy 2,3 i 4),
- c) *Pipeline Match* – dopasowywanie na podstawie pól dodanych do pakietu, innych niż nagłówek pakietu. Te pola to IN_PORT (wejściowy port), IN_PHY_PORT (wejściowy port fizyczny), METADATA (metadane przekazywane między tabelami) oraz TUNNEL_ID (metadane portu logicznego) [4].

Gdy rekord tabeli przepływu zostanie dopasowany, polecenia zawarte w polu *Instructions* zaczynają być wykonywane. Każde pole *Instructions* zawiera zbiór instrukcji (*Instruction set*), a każda instrukcja może, (lecz nie musi) zawierać listę akcji (*Action List*). Dopiero akcja oznacza jakieś konkretne polecenie. Opisane relacje w dobry sposób zilustrowane są na poniższym rysunku.



Rysunek 11: Relacje instrukcji i akcji [4].

Pole *Instruction* ze zbioru instrukcji (*Instruction Set*) może być typu:

- Meter *meter-id* (Opcjonalne) – wysyła pakiet do *meter table* wyspecyfikowanej przez *meter-id*,
- Apply-Actions *actions(s)* (Wymagane) – Wykonuje podaną listę akcji, bez zmieniania zbioru akcji,
- Clear-Actions (Opcjonalne) – usuwa wszystkie akcje ze zbioru akcji (*action set*),
- Write-Actions *action(s)* (Wymagane) – dodaje listę akcji do zbioru akcji,
- Write-Metadata *metadata/mask* (Opcjonalne) – zapisuje metadane do pola metadanych dla *pipeline processing*,
- Goto-Table *next-table-id* (Wymagane) – przejście do następnej tabeli przepływu [4].

W zbiorze instrukcji, każda z nich jest wykonywana w takiej kolejności, w jakiej zostały wymienione w powyższej liście. Dodatkowe określenie, które nie zostało do tej pory objaśnione to zbiór akcji (*action set*). Jest on inicjalizowany na początku *pipeline processing* i składa się z list akcji dodawanych przez instrukcję Write-Actions. Wszystkie listy ze zbioru akcji wykonywane są w momencie osiągnięcia przez pakiet ostatniej tabeli przepływu.

Po długim wstępie można w końcu wyjaśnić, w jaki sposób dodawane są operacje wyjściowe dla dopasowanych już pakietów w przełącznikach OpenFlow. Dzieje się to za pomocą wykonania akcji. Mogą one specyfikować wyjściowy port (*OUTPUT port*),

dodawać nowe pole VLAN (`PUSH_VLAN ethertype`) i wykonywać inne akcje, które zostaną opisane dokładniej później.

4.1.2. OVSDB

Specyfikacja tego protokołu jest tworzona m.in. w dokumencie *The Open vSwitch Database Management Protocol* przez organizację IETF. Dokument ten mówi o tym, w jaki sposób protokół porozumiewa się z bazą danych przełącznika OVS w celu jego konfiguracji. Ostatnia wersja dokumentacji została stworzona w Październiku 2014 roku.

Jak to było wcześniej wspomniane, switch OVS zawiera bazę danych (*ovsdb-server*), daemon'a urządzenia (*ovs-vswitchd*) i opcjonalnie moduł wykonujący szybkie przekierowywanie ścieżek, czyli moduł OpenFlow. Operacje wykonywane przez protokół OVSDB to:

- a) Tworzenie, modyfikacja i usuwanie wirtualnych mostków (*bridges*), których może być wiele w pojedynczej instancji OVS,
- b) Konfiguracja połączeń z kontrolerami dla każdego z mostków (według dokumentacji inna nazwa na mostek to *OpenFlow datapath*),
- c) Konfiguracja menadżerów, do których podpięty jest serwer OVSDB,
- d) Tworzenie, modyfikacja i usuwanie portów na mostkach,
- e) Tworzenie, modyfikacja i usuwanie interfejsów tunelowych (np. GRE) dla mostków,
- f) Tworzenie, modyfikacja i usuwanie kolejek,
- g) Konfiguracja polityki QoS i podpinanie jej do wcześniej zdefiniowanych kolejek,
- h) Zbieranie statystyk [6].

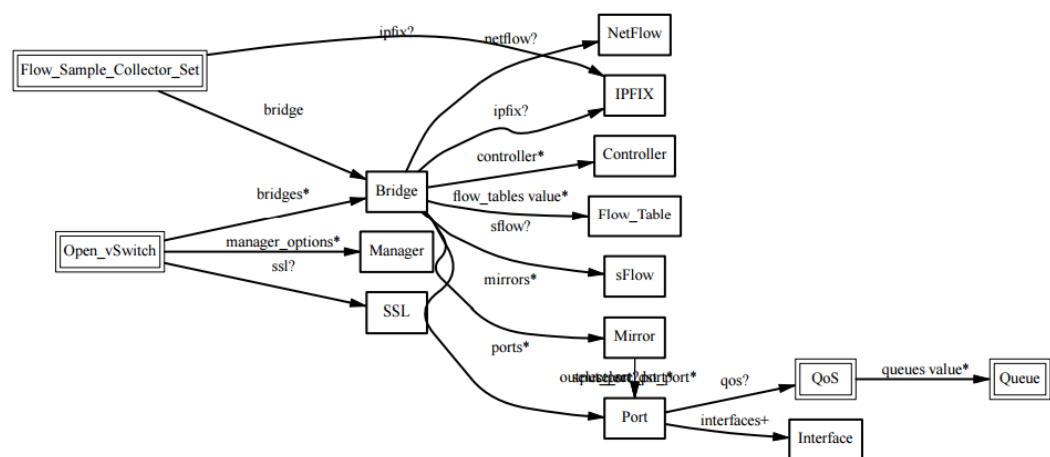
OVSDB używa formatu JSON, jako schemat danych i przy przekazywaniu danych, bazuje na JSON-RPC 1.0. We wspomnianym dokumencie oprócz opisanych już informacji znajduje się opis metod RPC i działania na bazie danych [6].

Oprócz powyżej opisanego dokumentu, warto zaznajomić się ze schematem bazy danych urządzenia OVS [9]. Tabele tam zawarte to:

- a) `Open_vSwitch` – konfiguracja OVSa,
- b) `Bridge` – konfiguracja mostka,
- c) `Port` – konfiguracja portu,
- d) `Interface` – fizyczne urządzenie sieciowe podpięte pod port,

- e) Flow_table – konfiguracja tabeli przepływów,
- f) QoS – konfiguracja jakości usługi,
- g) Queue – wyjściowa kolejka QoS,
- h) Mirror – port mirroring,
- i) Controller – konfiguracja kontrolera OpenFlow,
- j) Manager – konfiguracja połączenia OVSDb,
- k) NetFlow – konfiguracja NetFlow,
- l) SSL – konfiguracja SSL,
- m) sFlow – konfiguracja sFlow,
- n) IPFIX – konfiguracja IPFIX,
- o) Flow_Sample_Connector_Set – konfiguracja Flow_Sample_Connector_Set.

Rysunek 12 ilustruje relacje pomiędzy tymi tabelami.



Rysunek 12: Relacje pomiędzy tabelami z bazy danych OVS [9].

W dalszej części dokumentu opisywane są wszystkie wymienione tabele. Wybrane z nich będą opisane w dalszych podpunktach w zależności od potrzebnych informacji.

4.2. Zestawienie protokołów na podstawie dokumentacji

4.2.1. Tagowanie VLAN

a) OpenFlow

Operowanie na tagu VLAN w OpenFlow jest możliwe od wersji 1.1.0 i jest wykonywane za pomocą trzech akcji:

- *PUSH_VLAN ethertype* – Dodaje nowy nagłówek VLAN do pakietu, jeżeli tag już istnieje, nowy zostaje najbardziej zewnętrznym tagiem. Atrybut *ethertype* musi wynosić 0x8100 lub 0x88a8, czyli oznacza odpowiednio tagi C-TAG lub S-TAG według standardu MEF,
- *POP_VLAN* – Ściąga najnowszy tag VLAN,
- *SET_FIELD field* – Akcja ustawia wartości dla najbardziej zewnętrznego pola z nagłówka pakietu wyspecyfikowanego przez nazwę pola (*field*).
W tym przypadku jest to albo identyfikator VLAN (*VLAN_VID*) lub priorytet (*VLAN_PCP*) [4].

Przy założeniu, że używany przełącznik w pełni implementuje *OpenFlow Switch Specification* w wersji nie niższej niż 1.1.0, przy pomocy protokołu OpenFlow możemy dodawać, modyfikować i usuwać zarówno tagi VLAN dla klienta (C-TAG) jak i dla konkretnej usługi (S-TAG).

b) OVSDB

Konfiguracja VLAN w przypadku OVSDB może być ustawiana w tabeli *Port*. Operacje są wykonywane za pomocą tworzenia i modyfikacji wartości z następujących kolumn:

- *vlan-mode* – konfiguracja VLAN na porcie może być jednym z trzech typów: *access*, *native-tagged*, *native-untagged* lub *trunk*,
- *tag* – liczba całkowita od 0 do 4095,
- *trunks* – zbiór tagów, przy czym maksymalna ich liczba to 4096,
- *other_config: priority-tags* – wartość *true* lub *false*.

Działanie portu z ustawioną konfiguracją VLAN zależy od trybu konfiguracji, czyli od jednej z czterech wartości z kolumny *vlan-mode*:

- *trunk* – port trunk przepuszcza pakiety z nagłówkiem 802.1Q, których identyfikator VLAN znajduje się w kolumnie *trunks*, wszystkie inne pakiety są odrzucane.
- *access* – przepuszcza pakiety oznaczone VLAN tagiem ustawionym w kolumnie *tag*. Wszystkie inne są odrzucane.
- *native-tagged* – przypomina port *trunk*, z tą różnicą, że przychodzące pakiety bez nagłówka 802.1Q znajdują się w sieci *native VLAN*, o numerze

wyspecyfikowanym w kolumnie *tag* (dodawany jest nagłówek z danym VLAN ID),

- *native-untagged* – jest podobny do *native-tagged*, z tą różnicą, że pakiety wychodzące do sieci *native* nie są tagowane [9].

Podsumowując, jeśli przełącznik posiada zaimplementowaną funkcjonalność przedstawioną w schemacie bazy danych OVS, to protokół OVSDb potrafi tworzyć różne typy portów VLAN filtrujące ruch. Niestety, jeżeli chodzi o dodawanie nagłówków 802.1Q, OVSDb potrafi tylko w przypadku gdy port jest typu *native-tagged* i to w bardzo ograniczonym zakresie, podczas gdy OpenFlow nie posiada w tym temacie żadnych obostrzeń.

4.2.2. Przypisywanie pakietów do EVC

a) OpenFlow

Przypisywanie pakietów do konkretnej ścieżki jest głównym zadaniem OpenFlow, więc w tym przypadku OpenFlow nie posiada żadnych ograniczeń. Pakiety mogą być dopasowywane do konkretnych akcji na podstawie wielu parametrów. Przykładami mogą być port wejściowy (IN_PORT), adresy źródłowy (ETH_SRC, IPV4_SRC, IPV6_SRC) i docelowy (ETH_DST, IPV4_DST, IPV6_DST), identyfikator VLAN (VLAN_VID) i wielu innych.

Każdy dopasowany już pakiet, jeżeli nie został odrzucony, musi być przekierowany na jakiś port wyjściowy. W OpenFlow realizowane jest to za pomocą akcji „OUTPUT *port*”, gdzie *port* specyfikuje port wyjściowy. OpenFlow realizuje kryterium przypisywania pakietów do EVC zarówno w trybie *port-based* jak i *vlan-based*.

b) OVSDb

OVSDb nie jest protokołem służącym do przypisywania pakietów do konkretnych ścieżek, ma jednakże możliwość tworzenia tuneli za pomocą GRE (*Generic Routing Encapsulation*). Stworzenie takich tuneli odbywa się dzięki modyfikacji rekordów w tabeli *Interfaces*, która definiuje podpięty pod port interfejs. Kolumny dotyczące tunelowania to:

- *options : remote_ip* – Adres końcowy tunelu (IPv4),
- *options : local_ip* – Adres lokalny,

- *options : in_key* – klucz, który musi być zawarty w pakiecie wejściowym,
- *options : out_key* - klucz ustawiany przy wysyłaniu pakietów,
- *options : key* – skrót do ustawienia klucza wejściowego i wyjściowego za jednym razem,
- *options : tos* – wartość type-of-service,
- *options : ttl* – ustawiany czas życia enkapsulowanego pakietu,
- *options : df_default* – jeżeli ustawione na *true*, do nagłówka pakietu dodawane są bity *Don't fragment*,
- *options : csum* – obliczona suma kontrolna GRE dla wychodzących pakietów.

Aby stworzyć taki tunel, użytkownik musi tylko ustawić pole *remote_ip*, pozostałe są opcjonalne [9]. Dzięki stworzonemu tunelowi, każdy pakiet, który będzie przechodził przez dany port, zostanie przetransportowany na drugi jego koniec. Można to nazwać sposobem przypisania pakietu do EVC, ale na pewno nie efektywnym i wygodnym w konfiguracji. Tego typu tunele tworzy się raczej pomiędzy maszynami wirtualnymi, więc funkcjonalność tego rozwiązania pozostawia wiele do życzenia.

4.2.3. Definicja QoS

a) OpenFlow

W celu definicji jakości usługi za pomocą protokołu OpenFlow, trzeba skorzystać z wcześniej już wspomnianego komponentu *meter table*, wprowadzonego w wersji 1.3.0. Tabela ta pozwala na mierzenie szybkości przepływu i nakładanie odpowiednich ograniczeń. Każdy rekord tabeli *meter table* powiązany jest w rekordem z tabeli przepływu [4].

Pola z rekordu tabeli *meter* są następujące:

- *Meter identifier* – id reprezentowany przez 32 bitową liczbę całkowitą,
- *Meter Bands* – nieuporządkowana lista ograniczeń (*Meter Band*),
- *Counters* – Licznik, który jest zwiększany za każdym razem, kiedy przez tabelę przechodzi pakiet.

Pole *Meter Band* składa się z:

- *Band Type* – Definiuje jak pakiet ma być przetwarzany,

- *Rate* – Specyfikuje najwolniejszą szybkość jaką *meter band* może aplikować,
- *Burst* – definiuje ziarnistość *meter band*,
- *Counters* - licznik
- *Type Specific Arguments* – używany z niektórymi typami ograniczeń do podania dodatkowych instrukcji [4].

Przekierowanie do tabeli *meter* z rekordu tabeli przepływu następuje w chwili rozpoczęcia wykonywania instrukcji związanych z danym rekordem. Za pomocą *meter table* można ograniczyć szybkość przepływu pakietów, czyli możliwa w wykonaniu jest jedna z podstawowych funkcji QoS.

b) OVSDb

Definicję jakości usługi w przypadku OVSDb definiuje się za pomocą dodawania rekordów w tabeli QoS, która przypisywana jest do każdego portu. Posiada następujące pola:

- *Type* – typ implementowanego QoS. Na ten moment istnieje wsparcie dla *linux-htb* (Linux hierarchy token bucket classifier) oraz *linux-hfsc* (Linux "Hierarchical Fair Service Curve" classifier),
- *Queues* – mapa z rekordami w postaci (*integer, Queue*), gdzie drugi argument przedstawia tabelę kolejek,
- *other_config : max-rate* – określa maksymalną szybkość łącza. Domyślna wartość to 100 Mbps.

Tabela kolejek (*Queues*) wygląda następująco:

- *other_config : min-rate* – Minimalna gwarantowana przepustowość, w bit/s,
- *other_config : max-rate* – Maksymalna dopuszczalna przepustowość,
- *other_config : burst* – Maksymalna ilość „kredytu” podczas gdy kolejka jest bezczynna,
- *other_config : priority* – priorytet [9].

OVSDb ma dość spore możliwości, jeżeli chodzi o definicję QoS, ale niestety jest ograniczony do systemów opartych na linuxie, ponieważ mechanizmy definiujące jakość usługi bazują na *linux-htb* i *linux-hfsc*.

5. Zestawienie protokołów w rzeczywistych przypadkach użycia

W niniejszym podpunkcie stworzona zostanie wirtualna sieć komputerowa zawierająca przełączniki OVS, na których przeprowadzone zostaną działania konfiguracyjne za pomocą protokołów OpenFlow oraz OVSDb w celu ich porównania. Akcje te będą generowane poprzez dwa wybrane kontrolery SDN – OpenDayLight oraz ONOS.

5.1. Mininet

W celu porównania protokołów zasymulowana zostanie wirtualna sieć komputerowa za pomocą programu *mininet*. Wszystkie tworzone w ten sposób przełączniki są urządzeniami OVS. W tej pracy został użyty *mininet* w wersji 2.2.2, uruchomiony na wirtualnej maszynie Ubuntu 14.04, przygotowanej przez twórców tego oprogramowania.

Jednym z pierwszych kroków konfiguracyjnych jest ustalenie sposobu zdalnego zarządzania urządzeniami OVS na danej maszynie. Istnieją dwa sposoby:

- Aktywny Manager – OVS jest przyczepiany do działającej instancji kontrolera, który w tym przypadku nasłuchuje na danym porcie. Organizacja IANA zarejestrowała port 6640 dla OVS. Aby ustawić aktywnego zarządcę na dany kontroler, trzeba wywołać poniższą komendę na maszynie, na której działa OVS [12].

```
root@mininet-vm:~# ovs-vsctl set-manager tcp:127.0.0.1:6640
```

Listing 1: Ustawienie aktywnego menadżera OVS [źródło własne].

Przy czym adres IP zależy od adresu maszyny, na której znajduje się działający kontroler, w tym przypadku jest to ta sama maszyna.

- Pasywny Manager – W tym trybie, to kontroler inicjuje połączenie z nasłuchującym klientem urządzeń OVS [12]. Aby rozpocząć takie nasłuchiwanie, po stronie OVS trzeba wywołać polecenie z listingu 2.

```
root@mininet-vm:~# ovs-vsctl set-manager ptcp:6640
```

Listing 2: Ustawienie pasywnego menadżera OVS [źródło własne].

Status powyżej opisywanego połączenia można sprawdzić za pomocą komendy *ovs-vsctl show* (opisanej w następnym podrozdziale).


```

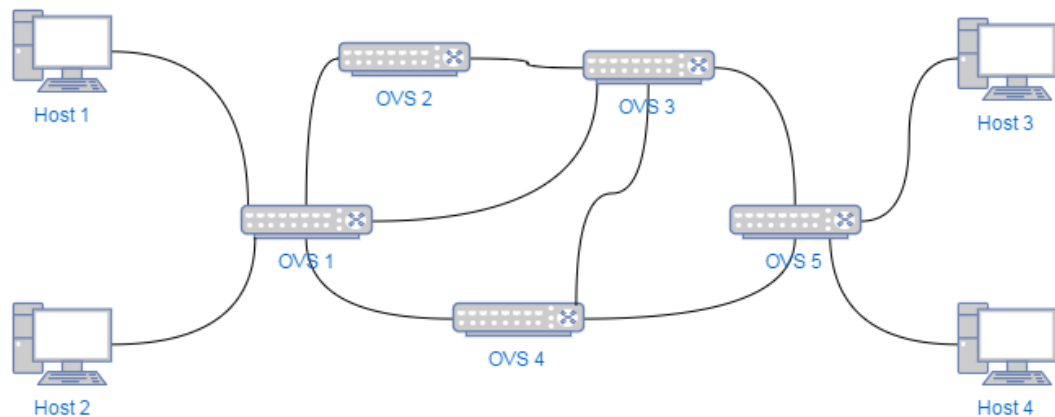
root@mininet-vm:~# ovs-vsctl show
df048a2d-88b3-4d49-85fc-c670bfd8b051
    Manager "ptcp:6640"
    is_connected: true
    ovs_version: "2.7.0"

```

Rysunek 13: Ustawione pasywne połączenie [źródło własne].

Jak widać na powyższym rysunku, manager urządzeń jest pasywny i nasłuchuje na porcie 6640. Flaga `is_connected: true`, oznacza, że kontroler nawiązał połączenie. W niniejszej pracy ustalony jest manager pasywny w celu łatwiejszego zmieniania kontrolerów zarządzających siecią wirtualnych przełączników OVS.

W niniejszej pracy to porównania protokołów wymyślona została topologia składająca się z czterech wirtualnych komputerów, podłączonych do dwóch krańcowych przełączników, pomiędzy którymi znajdują się 3 pośrednie przełączniki. Sieć ta przedstawiona jest na rysunku 14.



Rysunek 14: Topologia sieciowa [źródło własne]

W programie mininet topologię sieci definiuje się za pomocą skryptu napisanego w języku Python. Skrypt definiujący powyższą topologię wygląda następująco:

```

1 from mininet.topo import Topo
2
3 class MgrTopo( Topo ):
4     "Topology prepared for Master Thesis"
5
6     def __init__( self ):
7
8         # Initialize topology
9         Topo.__init__( self )
10
11        # Add hosts and switches
12        h1 = self.addHost( 'h1' )
13        h2 = self.addHost( 'h2' )
14        h3 = self.addHost( 'h3' )
15        h4 = self.addHost( 'h4' )
16

```

```

17         s1 = self.addSwitch( 's1' )
18         s2 = self.addSwitch( 's2' )
19         s3 = self.addSwitch( 's3' )
20         s4 = self.addSwitch( 's4' )
21         s5 = self.addSwitch( 's5' )
22
23         # Add links
24         self.addLink( h1, s1 )
25         self.addLink( h2 , s1 )
26         self.addLink( h3 , s5 )
27         self.addLink( h4 , s5 )
28
29         self.addLink( s1, s2 )
30         self.addLink( s1, s3 )
31         self.addLink( s1, s4 )
32         self.addLink( s2, s3 )
33         self.addLink( s3, s4 )
34         self.addLink( s3, s5 )
35         self.addLink( s4, s5 )

```

Listing 3: Definicja sieci mininet [źródło własne].

W powyższym przykładzie docelowa topologia została stworzona dzięki dziedziczeniu po domyślnej topologii mininet o nazwie *Topo* (linia 3) importowanej z pakietu *mininet.topo* w linii 1. Od linii 6 następuje definicja funkcji *init* odpowiadającej za określenie topologii sieci mininet. Od linii 12 do 16 znajduje się dołączenie wirtualnych komputerach o uproszczonych nazwach: *h1,h2,h3* i *h4*. W liniach 17-22 utworzone zostają wirtualne przełączniki o nazwach *s1,s2,s3,s4* i *s5*. Po definicji wirtualnych urządzeń w liniach od 24 do 27 następuje określenie połączenia switch'y z komputerami, a w liniach 29-35 połączenia między przełącznikami. Uruchomienie takiej topologii odbywa się za pomocą poniższego skryptu:

```

1 if __name__ == '__main__':
2     controller_ip = '127.0.0.1'
3
4     OVSSwitch13 = partial( OVSSwitch, protocols='OpenFlow13' )
5     topo=MgrTopo( )
6     net = Mininet( topo , switch=OVSSwitch13 ,
controller=partial( RemoteController, ip= controller_ip, port=6633 )
7 )
8     net.start()
9
10    CLI( net )
11    net.stop()

```

Listing 4: Skrypt uruchamiający sieć mininet [źródło własne].

W powyższym skrypcie następuje zdefiniowanie dodatkowych argumentów określających wirtualną sieć komputerową. W 4 linii tworzona jest definicja switch'a OVS, gdzie dzięki funkcji *partial* podawana jest wersja OpenFlow 1.3.0, co później jest wzięte pod uwagę przy tworzeniu wirtualnych przełączników. W 5 linii inicjalizowana

jest zdefiniowana w poprzednim skrypcie topologia. Linijka 5 zawiera definicje sieci wirtualnej, gdzie podajemy topologię, definicję OVS'a i definicję kontrolera SDN. W powyższym przypadku kontroler jest na tej samej maszynie co sieć, więc jego adres to adres lokalny 127.0.0.1, dodatkowo podajemy port, na którym rozmawiają kontroler z przełącznikami. W linii 8 rozpoczynane jest działanie sieci wirtualnej, po czym otwierana jest konsola programu mininet, służąca do konfiguracji urządzeń. W momencie zamknięcia konsoli, sieć jest wyłączana poprzez komendę `net.stop()`. Tak przygotowany skrypt uruchamia się za pomocą polecenia:

```
~/mgr# python test.py
```

Listing 5: Komenda uruchamiająca skrypt zdefiniowany w Listingu 2 [źródło własne].

Przy uruchamianiu skryptu mininet tworzy wirtualne przełączniki, komputery i połączenia między nimi oraz ustanawia połączenie z kontrolerem SDN. Po wszystkim uruchamiana jest konsola programu, dzięki której można bezpośrednio konfigurować wszystkie wirtualne urządzenia. Proces uruchamiania jest zwizualizowany na rysunku

```
root@mininet-vm:~/mgr# python create_mininet_topology.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s1) (h3, s5) (h4, s5) (s1, s2) (s1, s3) (s1, s4) (s2, s3) (s3, s4) (s3, s5) (s4, s5)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> sh ifconfig
```

Rysunek 15: Start programu mininet [źródło własne].

Wirtualne komputery emulują działanie systemu, na jakim działa mininet, więc można je konfigurować za pomocą takich samych komend jak w systemie, na którym uruchomiony jest mininet. Przykładowo, w celu zobaczenia konfiguracji pierwszego z emulowanych komputerów trzeba wykonać następującą komendę w konsoli:

```
mininet> h1 ifconfig
```

Listing 6: Komenda wypisująca ustawienia sieciowe wirtualnego komputera h1 [źródło własne].

Oprócz tego, przydatną komendą jest *ping*, dzięki, którym można sprawdzić komunikację pomiędzy wirtualnymi komputerami, poprawne odpowiedzi świadczą o dobrej inicjalizacji reguł OpenFlow na switch'ach. Z konsoli mininet do linii poleceń

systemu operacyjnego można się dostać poprzez dodanie przedrostka *sh* przed danym poleceniem.

5.2. Dostępne narzędzia do konfigurowania przełączników OVS

Wirtualne przełączniki OVS można konfigurować głównie za pomocą opisywanych protokołów OpenFlow i OVSDb. Aby móc z nich jednak korzystać potrzebne są odpowiednie do tego narzędzia. Najbardziej znane narzędzia to:

- a) Kontrolery SDN – Opisywane w niniejszej pracy, najczęściej posiadają własny model danych odwzorowujący informacje zawarte w wirtualnych przełącznikach. Konkretnie kontrolery i ich sposób połączenia ze switch'ami OVS zostanie opisany w kolejnych podrozdziałach,
- b) OVN (*Open Virtual Network*) - Wolno dostępny projekt stworzony przez zespół tworzący OVS. Jego celem jest stworzenie ujednoliconego, niezależnego od dostawcy protokołu służącego do wirtualizacji funkcji sieciowych oferowanych przez różnego rodzaju przełączniki. Dzięki temu różni klienci mogą korzystać z danych funkcji sieciowych za pomocą ujednoliconego API. OVN może używać protokołów OpenFlow oraz OVSDb do dostarczania funkcjonalności z warstwy 2 i 3 modelu OSI/ISO [10].
- c) Programy dostarczane podczas instalacji OVS. Używa się ich z linii poleceń. Najbardziej znane to:
 - Ovs-vsctl – Program dostarcza interfejs do konfiguracji bazy danych urządzenia OVS. Umożliwia dodawanie, modyfikacje i usuwanie switch'ów, portów i interfejsów na portach. Ponadto umożliwia ustawianie ograniczenia szybkości przepływu pakietów i wiele innych akcji zdefiniowanych w schemacie bazy danych.
 - Ovs-ofctl – Zarządzanie zadaniami związanymi z OpenFlow. Przykładowe komendy to:
 - `show switch` – wypisanie informacji związanych z OpenFlow na danym switch'u,
 - `add-flow switch flow` – Dodanie reguły OpenFlow (*flow*) do przełącznika (*switch*),
 - `dump-flows switch` – Wypisanie wszystkich reguł przepływu na danym przełączniku,

- del-flow *switch flow* – usunięcie reguły na danym przełączniku

5.3. OpenDayLight

5.3.1. Opis

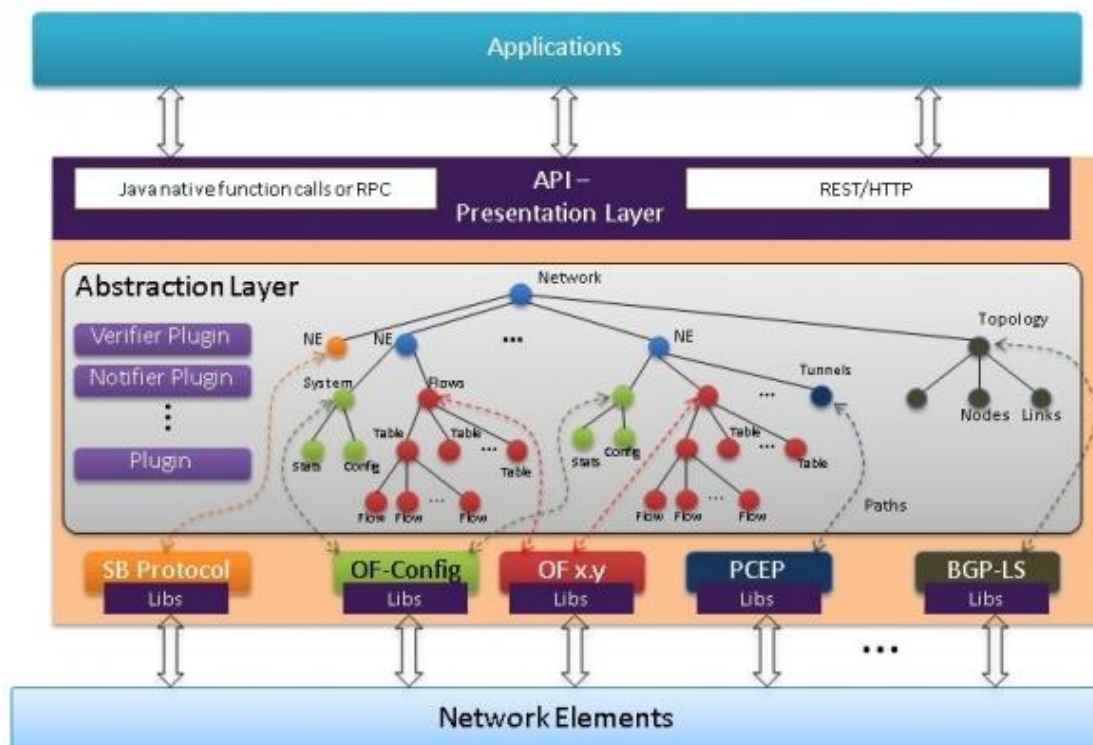
OpenDayLight (ODL) jest jednym z najlepszych kontrolerów SDN wśród wszystkich wolno dostępnych systemów tego typu, obecnie dostępnych na rynku sieci komputerowych. Organizacja *Linux Foundation* w Kwietniu 2013 roku ogłosiła powstanie projektu *OpenDayLight*, w celu stworzenia bardziej otwartego i klarownego podejścia do sieci programowalnych (SDN) oraz wirtualizacji funkcji sieciowych (NFV). Projekt jest finansowany i rozwijany przez największych graczy w branży sieci komputerowych, takich jak Cisco, Ericsson, IBM, VMWare i wielu innych [11].

ODL jest dostępny na licencji *Eclipse Public License (EPL-1.0)*. System jest napisany w języku Java i działa na kontenerze modułów OSGi (*bundle*) – Karafie, co pozwala na dodawanie danych funkcjonalności w czasie działania oprogramowania bez potrzeby zatrzymywania lub resetowania kontrolera. W terminologii ODL narzuconej przez wykorzystanie kontenera Karaf, moduły dostarczające daną funkcjonalność posiadają nazwę *feature*, która będzie wykorzystywana częściej w tym podrozdziale.

Architekturę ODL na wysokim poziomie abstrakcji można podzielić na trzy główne części:

- Northbound interfaces – Interfejsy *północne* kontrolera odpowiadający za połączenie z innymi aplikacjami, np. z orkiestratorem sieciowym (programem często wykorzystywanym przez dostawców usług sieciowych). Dzieje się to zazwyczaj za pomocą zwołać Java RPC, lub protokołów REST, lub innych opratych na HTTP.
- Southbound interfaces – Interfejsy *południowe*, odpowiedzialny za komunikację z urządzeniami zarządzanymi. Najczęściej są to moduły obsługujące dany protokół komunikacyjny, np. OpenFlow, OVSDb, NetConf itd.
- Controller logic – Logika kontrolera, której zadaniem jest pośredniczenie pomiędzy interfejsami północnymi i południowymi, dostarczając przy tym jakąś wartość dodaną, np. wybierając odpowiedni moduł do konfiguracji

danego urządzenia sieciowego. W tym miejscu znajduje się również informacja o całej zarządzanej przez kontroler topologii sieciowej. Opisywaną architekturę reprezentuje załączony poniżej rysunek.



Rysunek 16: Architektura kontrolera ODL [12].

ODL posiada swoje nazwy od kolejnych pierwiastków w układzie okresowym. Pierwszą wersją był *Hydrogen* wydany Lutym 2014, najnowszą wersję reprezentuje *Carbon* wydany w Czerwcu 2017 [11]. W niniejszej pracy wykorzystywany będzie kontroler ODL w wersji najnowszej.

Rzeczą, o której trzeba napisać omawiając ten kontroler jest język YANG i jego korelacja z ODL. YANG (*Yet Another Next Generation*) jest językiem służącym do modelowania danych przesyłanych poprzez protokół NETCONF (*Network Configuration Protocol*). Może być użyty do modelowania:

- danych konfiguracyjnych,
- operacyjnych (wskazujących obecny stan urządzenia sieciowego),
- formatu notyfikacji emitowanych przez urządzenia sieciowe [13].

Przechowywanie danych w ODL jest zrealizowane za pomocą komponentu MD-SAL (*Model Driven – Service Abstraction Layer*). Główne cele tego komponentu to:

- Definicja wspólnych pojęć, modelu danych i wzorców dostarczających infrastrukturę do komunikacji w aplikacji oraz między aplikacjami,
- Zapewnienie wsparcia dla formatów danych definiowanych przez użytkowników.

Aby osiągnąć powyższe cele, MD-SAL używa YANG'a jako języka do modelowania interfejsów i danych. Dane w ODL przechowywane są w dwóch drzewach (*Data Tree*):

- *Operational Data Tree* – Zawiera stan systemu, uaktualniany przez dostawców informacji (urządzenia sieciowe, aplikacje itd.) poprzez MD-SAL,
- *Configuration Data Tree* – Przewidywany stan systemu, aktualizowany przez użytkownika [12].

Po zainstalowaniu modułu *odl-restconf-all*, można przeglądać i nadpisywać opisane powyżej drzewa za pomocą protokołu RESTCONF. Czyli protokołowi podobnemu do REST, korzystającemu z HTTP w celu dostępu do danych zdefiniowanych przez YANG.

5.3.2. Uruchomienie kontrolera i połączenie z siecią mininet

Po ściągnięciu najnowszej dystrybucji OpenDayLight, został on rozpakowany na lokalnym komputerze z systemem Windows 7. Jedynym niezbędnym warunkiem potrzebnym do uruchomienia tego kontrolera jest zainstalowane JRE (*Java Runtime Environment*). Przy tak przygotowanym środowisku, instancję kontrolera powinno uruchamiać się z folderu *bin*:

```
${odl_install_path}\distribution-karaf-0.6.0-Carbon\bin
```

Listing 7: Ścieżka do folderu bin [źródło własne].

Będąc w tym folderze, kontroler ODL można uruchomić za pomocą skryptów *karaf.bat* lub *start.bat* w zależności od tego czy potrzebujemy mieć dostęp do linii poleceń instancji ODL (w przypadku uruchomienia poprzez drugi skrypt, do klienta tekstowego kontrolera można się dostać za pomocą skryptu *client.bat*). Po uruchomieniu skryptu, pojawić się powinien ekran startowy i linia poleceń, widoczne na poniższym rysunku.

```
PS C:\ODL\distribution-karaf-0.6.0-Carbon\bin> .\karaf.bat
Apache Karaf starting up. Press Enter to open the shell now...
100% [=====]

Karaf started in 7s. Bundle stats: 64 active, 64 total

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Rysunek 17: Uruchomienie kontrolera OpenDayLight [źródło własne].

Dzięki wsparciu kontenera *karaf*, dla komend z powłoki *bash*, istnieje możliwość wypisania interesujących nas funkcjonalności za pomocą kombinacji dwóch komend: *feature:list* i *grep*. Kolejne polecenia scala się za pomocą tzw. *pipe* „|”, tak samo jak powłoce *bash*. W celu znalezienia dostępnych funkcjonalności związanych z protokołem OVSDb należy wykonać komendę widoczną na rysunku 19.

```
opendaylight-user@root>feature:list | grep ovbdb
odl-ovbdb-library | 1.4.0-Carbon | x | odl-ovbdb-library-1.4.0-Carbon
| OpenDaylight :: library | 1.4.0-Carbon | | odl-ovbdb-hwtepsouthbound-1.4.0-Carbon
n | OpenDaylight :: hwtepsouthbound :: api | 1.4.0-Carbon | | odl-ovbdb-hwtepsouthbound-1.4.0-Carbon
n | OpenDaylight :: hwtepsouthbound | 1.4.0-Carbon | | odl-ovbdb-hwtepsouthbound-1.4.0-Carbon
odl-ovbdb-hwtepsouthbound-rest | 1.4.0-Carbon | | odl-ovbdb-hwtepsouthbound-1.4.0-Carbon
n | OpenDaylight :: hwtepsouthbound :: REST | 1.4.0-Carbon | | odl-ovbdb-hwtepsouthbound-1.4.0-Carbon
n | OpenDaylight :: hwtepsouthbound :: UI | 1.4.0-Carbon | | odl-ovbdb-hwtepsouthbound-1.4.0-Carbon
odl-ovbdb-hwtepsouthbound-test | 1.4.0-Carbon | | odl-ovbdb-hwtepsouthbound-1.4.0-Carbon
n | OpenDaylight :: hwtepsouthbound :: test | 1.4.0-Carbon | x | odl-ovbdb-southbound-1.4.0-Carbon
| OpenDaylight :: southbound :: api | 1.4.0-Carbon | x | odl-ovbdb-southbound-1.4.0-Carbon
| OpenDaylight :: southbound :: impl | 1.4.0-Carbon | x | odl-ovbdb-southbound-1.4.0-Carbon
odl-ovbdb-southbound-impl-rest | 1.4.0-Carbon | | odl-ovbdb-southbound-1.4.0-Carbon
| OpenDaylight :: southbound :: impl :: REST | 1.4.0-Carbon | | odl-ovbdb-southbound-1.4.0-Carbon
| OpenDaylight :: southbound :: impl :: UI | 1.4.0-Carbon | | odl-ovbdb-southbound-1.4.0-Carbon
odl-ovbdb-southbound-test | 1.4.0-Carbon | | odl-ovbdb-southbound-1.4.0-Carbon
| OpenDaylight :: southbound :: test
```

Rysunek 18: Wypisanie dostępnych funkcjonalności związanych z OVSDb w kontrolerze ODL [źródło własne].

Zainstalowane moduły zaznaczone są za pomocą litery *x*. W celu dodania nowej funkcjonalności, należy wykonać polecenie *feature:install x_feature*, gdzie *x_feature* to nazwa modułu. W celu porównania protokołów zainstalowane zostały wszystkie moduły związane z OpenFlow oraz OVSDb.

W celu ustawienia kontrolera ODL, jako zdalnego menedżera urządzeń OVS, niezbędną akcją jest dodanie węzła w topologii OVSDb reprezentującego połączenie z klientem OVS zarządzającym wirtualnymi przełącznikami. W kontrolerze ODL można to zrobić dzięki użyciu protokołu RESTCONF, poprzez wysłanie zapytania POST na adres *http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/* z poniższym ciałem.

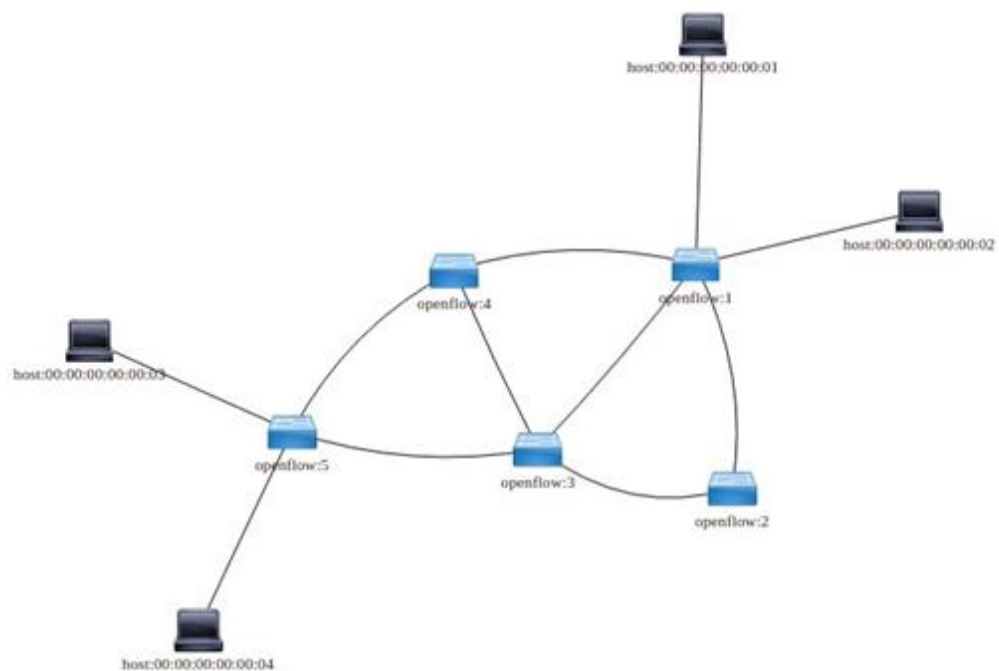

```

1 {
2     "node": [
3         {
4             "node-id": "mgr",
5             "connection-info": {
6                 "remote-ip": "192.168.8.106",
7                 "remote-port": 6640
8             }
9         }
10    ]
11 }

```

Listing 8: Ustawienie połączenia z urządzeniami OVS [źródło własne].

Powyżej w linii 4 ustawiany jest identyfikator węzła, w 6 adres IP urządzenia, na którym znajduje się wirtualna sieć urządzeń OVS i w 7 port na którym nasłuchuje. Po dodaniu takiego węzła, w topologii *Operational* powinny pojawić się wszystkie wirtualne przełączniki OVS wraz ze wszystkimi portami i interfejsami z dodanego urządzenia. Po zainstalowaniu modułów odpowiedzialnych za interfejs graficzny (*odl-dlux-core* oraz *odl-dluxapps-topology*) możliwa jest wizualizacja stworzonej topologii, poprzez wejście na stronę <http://localhost:8181/index.html#/topology> (Rysunek 20).



Rysunek 19: Topologia mininet zwizualizowana za pomocą ODL [źródło własne].

Na powyższej topologii obowiązuje nomenklatura OpenFlow.

5.3.3. Analiza OpenFlow i OVSDb w kontrolerze

Komunikacja kontrolera z przełącznikami OVS odbywa się za pomocą tzw. *southbound plugins*. Są to komponenty funkcyjne zapewniające abstrakcyjny widok funkcjonalności urządzeń sieciowych, normalizujące ich API i jest odpowiedzialny za utrzymanie z nimi połączenia. Abstrakcyjny model jest tworzony przez definicję YANG, które są później parsowane do klas Java umożliwiając implementację plugina [12].

W niniejszej pracy zainstalowane zostały poniższe moduły:

- *odl-openflowjava* – Zestaw interfejsów i ich implementacji niezmiennych obiektów danych reprezentujących struktury protokołu Openflow [12].
- *odl-openflowplugin* – Moduł ten jest odpowiedzialny za implementację działania protokołu OpenFlow. Aktualnie wspierana jest specyfikacja OpenFlow w wersjach od 1.0 do 1.3.x,
- *odl-ovsdb* – Moduł dostarcza plugin do zarządzania urządzeniami OVS, przez protokół OVSDb [12].

Oprócz modułów odpowiedzialnych za konfigurację urządzeń na podstawie poleceń użytkownika, przydatne są moduły, które konfiguruja urządzenia OVS bez konkretnych poleceń. Przykładem jest *odl-openflowplugin-app-forwardingrules*, który inicjalizuje reguły OpenFlow w przełącznikach OVS w momencie połączenia wirtualnej sieci do kontrolera ODL. Dzięki takiej inicjalizacji możliwa jest komunikacja pomiędzy wirtualnymi komputerami w przygotowanej topologii przedstawionej we wcześniejszych rozdziałach.

```
mininet> h1 ping -c 3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.596 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.392 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.271 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.271/0.419/0.596/0.136 ms
mininet>
```

Rysunek 20: Zaprezentowane połączenie pomiędzy wirtualnymi komputerami [źródło własne].

W tak skonfigurowanym zestawieniu można przedstawić sposoby konfiguracji urządzeń OVS w kontrolerze OpenDayLight. Zmian dokonuje się poprzez zmiany wprowadzone w magazynie danych ODL (*DataStore*). Dokonać tego można

za pomocą wcześniej wspomnianego protokołu RESTCONF. Aby poznać schemat magazynu danych ODL, warto zapoznać się z dokumentacją modelu danych MD-SAL. Dostępna jest ona na stronie <http://localhost:8181/apidoc/explorer/index.html>, po zainstalowaniu modułu *odl-mdsal-apidocs*.

lldp-speaker(2014-10-23)	Show/Hide List Operations Expand Operations Raw
lldp-speaker-config(2016-05-12)	Show/Hide List Operations Expand Operations Raw
loop-remover-config(2014-05-28)	Show/Hide List Operations Expand Operations Raw
nc-notifications(2008-07-14)	Show/Hide List Operations Expand Operations Raw
network-topology(2013-07-12)	Show/Hide List Operations Expand Operations Raw
network-topology(2013-10-21)	Show/Hide List Operations Expand Operations Raw
POST /config	
GET /config/network-topology:network-topology	
PUT /config/network-topology:network-topology	
DELETE /config/network-topology:network-topology	
POST /config/network-topology:network-topology	
<div>Response Class</div> <div>Model Model Schema</div> <pre>(config)network-topologyPOST { topology (array[network-topology/network-topology(config)topology], optional) } network-topology/network-topology(config)topology { network-topology:underlay-topology (array[network-topology/network-topology/topology(config)underlay-topology], optional): Identifies the topology, or topologies, that this topology is dependent on, network-topology:link (array[network-topology/network-topology/topology(config)link], optional): A Network Link connects a by Local (Source) node and a Remote (Destination) Network Nodes via a set of the nodes' termination points. As it is possible to have several links between the same source and destination nodes, and as a link could potentially be re-homed between termination points, to ensure that we would always know to distinguish between links, every link is identified by a dedicated link identifier. Note that a link models a point-to-point link, not a multipoint link. Layering dependencies on links in underlay topologies are not represented as the layering information of nodes and of termination points is sufficient. network-topology:topology-id (Some topology-id, optional): It is presumed that a datastore will contain many topologies. To distinguish between topologies it is vital to have UNIQUE topology identifiers. }</pre>	

Rysunek 21: Dokumentacja MD-SAL [źródło własne].

Jak widać na powyższym rysunku, dzięki takiej dokumentacji można dowiedzieć się, jakie operacje na danych są dostępne i w jaki sposób ich używać. Dodatkowym atutem tej jest możliwość wysyłania zapytań RESTCONF bezpośrednio ze strony www. Podczas wykonywania tej pracy używane będzie narzędzie POSTMAN, służące do wykonywania zapytań REST. Decyzja ta podyktowana jest tym, że istnieje możliwość definicji kolekcji zapytań, które można w prosty i szybki sposób wielokrotnie używać w odróżnieniu od interfejsu graficznego na stronie apidocs, gdzie akcje wykonują się wolniej i nie ma możliwości zapisania ulubionych parametryzowanych zapytań.

Dane konfiguracyjne dotyczące OpenFlow znajdują się w *opendaylight-inventory*. Przykładowo, w momencie, gdy użytkownik chce uzyskać informację

na temat reguł OpenFlow przełącznika *s1* (*openflow:1* w nomenklaturze OpenFlow) w domyślnej tabeli 0, musi wykonać zapytanie GET na adresie *http://localhost:8181/restconf/operational/.opendaylight-inventory:nodes/node/openflow:1/table/0*. Wynik tak wykonanego zapytania zaprezentowany jest poniżej.

```
1 {
2     "flow-node-inventory:table": [
3         {
4             "id": 0,
5             "flow": [
6                 {
7                     "id": "L2switch-4",
8                     "cookie_mask": 0,
9                     "priority": 2,
10                    "table_id": 0,
11                    "opendaylight-flow-statistics:flow-statistics":
12                    {
13                        "duration": {
14                            "second": 16046,
15                            "nanosecond": 188000000
16                        },
17                        "byte-count": 0,
18                        "packet-count": 0
19                    },
20                    "match": {
21                        "in-port": "3"
22                    },
23                    "flags": "",
24                    "idle-timeout": 0,
25                    "hard-timeout": 0,
26                    "cookie": 3098476543630901252,
27                    "instructions": {
28                        "instruction": [
29                            {
30                                "order": 0,
31                                "apply-actions": {
32                                    "action": [
33                                        {
34                                            "order": 2,
35                                            "output-action": {
36                                                "output-node-
37                                                    "max-length": 65535
38                                            }
39                                        },
40                                        {
41                                            "order": 1,
42                                            "output-action": {
43                                                "output-node-
44                                                    "max-length": 65535
45                                            }
46                                        }
47                                    ]
48                                }
49                            }
50                        ]
51                    }
52                }
53            ]
54        }
55    ]
56 }
```

Listing 9: Stan tabeli 0 z przełącznika *openflow:1* [źródło własne].

Jak widać na listingu 9 przedstawiona jest część stanu tabeli reguł OpenFlow z domyślnej tabeli o id=0 po inicjalizacji poprzez jeden z modułów ODL. Powyżej przedstawiona jest reguła, która dopasowuje wszystkie pakiety przychodzące na port 3 (linia 20). Akcje wyjściowe to przekierowanie pakietów na porty 1 (linia 35) i 2 (linia 42).

Dane dotyczące OVSDb znajdują się w drzewie o nazwie *network-topology*, w topologii o nazwie *ovsdb:1*. Takie dane można otrzymać poprzez wykonanie zapytania GET na następującym adresie URL - *http://localhost:8181/restconf/operational/network-topology:network-topology/topology/ovsdb:1/*.

5.3.3.1. Tagowanie VLAN

a) OpenFlow

W celu dodania reguł OpenFlow poprzez ODL, należy je dodać do drzewa *opendaylight-inventory:nodes* z przedrostkiem *config*, ponieważ jedynie w ten sposób moduł *odl-openflowplugin-southbound* może zaaplikować podane reguły na rzeczywistych urządzeniach. W starszych wersjach ODL trzeba było definiować wersję OpenFlow przy uruchamianiu kontenera karaf poprzez dodanie flagi np. *-of13*. Obecnie domyślnie wspierana jest wersja OpenFlow 1.3.

Przykładowo w celu dodania reguły OpenFlow o id *vlan4* do przełącznika o id *openflow:1*, do tabeli 0, należy wykonać metodę PUT na następującym adresie URL - *http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/vlan4*, z ciałem metody przedstawionym na listingu 10.

```
1 {
2   "flow-node-inventory:flow": [
3     {
4       "id": "vlan4",
5       "priority": 10,
6       "table_id": 0,
7       "opendaylight-flow-statistics:flow-statistics": {
8         "duration": {
9           "second": 16307,
10          "nanosecond": 828000000
11        },

```

```

12         "byte-count": 2822,
13         "packet-count": 43
14     },
15     "match": {
16         "vlan-match": {
17             "vlan-id": {
18                 "vlan-id-present": true,
19                 "vlan-id": 1234
20             }
21         },
22         "in-port": "openflow:1:2"
23     },
24     "flags": "",
25     "idle-timeout": 0,
26     "hard-timeout": 0,
27     "cookie": 3098476543630901312,
28     "instructions": {
29         "instruction": [
30             {
31                 "order": 0,
32                 "apply-actions": {
33                     "action": [
34                         {
35                             "order": 0,
36                             "pop-vlan-action": {}
37                         },
38                         {
39                             "order": 1,
40                             "output-action": {
41                                 "output-node-connector": "1",
42                                 "max-length": 65535
43                             }
44                         },
45                         {
46                             "order": 3,
47                             "output-action": {
48                                 "output-node-connector": "3",
49                                 "max-length": 65535
50                             }
51                         },
52                         {
53                             "order": 2,
54                             "output-action": {
55                                 "output-node-connector": "4",
56                                 "max-length": 65535
57                             }
58                         }
59                     ]
60                 }
61             }
62         ]
63     }
64 }
65 ]
66 }

```

Listing 10: Dodanie reguły VLAN [źródło własne].

Na powyższym listingu przedstawiona jest reguła, która dodaje dopasowanie pakietu nie tylko na podstawie portu, przez który pakiet wszedł (linia 22), ale również na podstawie tagu VLAN (linia 19). Po dopasowaniu pakietu reguła ściąga tag VLAN (linia 36), a następnie przekierowuje pakiety na porty 1,3 i 4.

W celu dodania tagu VLAN, do analogicznego ciała metody PUT do powyższej, trzeba dodać poniżej zdefiniowane akcje.

```
1      {
2          "order": 0,
3          "push-vlan-action": {
4              "ethernet-type": 33024
5          }
6      },
7      {
8          "order": 1,
9          "set-field": {
10             "vlan-match": {
11                 "vlan-id": {
12                     "vlan-id" : "1234",
13                     "vlan-id-present": "true"
14                 }
15             }
16         }
17     }
```

Listing 11: Dodanie akcji dodania tagu vlan, [[źródło własne]].

W listingu 11, pierwsza akcja dodaje pole VLAN do pakietu, a następna inicjalizuje je z identyfikatorem 1234.

Podsumowując, dzięki kontrolerowi ODL można wykonywać wszystkie akcje zdefiniowane w wersji OpenFlow 1.3, czyli dopasowywanie pakietów po tagu VLAN, oraz dodawanie tego tagu.

b) OVSDB

ODL posiada model bazy danych OVSDB, dzięki czemu przy użyciu modułu *odl-ovsdb-southbound*, istnieje możliwość konfiguracji tabel bazy danych OVSDB. Funkcje te zostały przedstawione w podrozdziale 4.2.1, w punkcie b. Przykładowa modyfikacja bazy OVSDB poprzez metodę PUT na adres `http://<controller-ip>:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:%2F%2FHOST1%2Fbridge%2Fbr-test/termination-point/testport/`, znajduje się na poniższym listingu.

```

1 {
2   "network-topology:termination-point": [
3     {
4       "ovsdb:options": [
5         {
6           "ovsdb:option": "remote_ip",
7           "ovsdb:value" : "10.10.14.11"
8         }
9       ],
10      "ovsdb:name": "testport",
11      "ovsdb:interface-type": "ovsdb:interface-type-vxlan",
12      "tp-id": "testport",
13      "vlan-tag": "1",
14      "trunks": [
15        {
16          "trunk": "5"
17        }
18      ],
19      "vlan-mode": "access"
20    }
21  ]
22 }

```

Listing 11: Dodanie interfejsu VLAN [13].

W powyższym listingu do portu *testport* (linia 10) dodawany jest interfejs VLAN (linia 11), typu *access* (linia 19). Powyższa konfiguracja oznacza, że dopasowywane są pakiety o id tagu vlan równym 5 (linia 6) i na wyjściu z przełącznika nadawany jest nowy vlan id równy 1 (linia 13).

5.3.3.2. Przypisywanie pakietów do EVC

W ODL do tworzenia usług zdefiniowanych przez MEF istnieje moduł o nazwie *odl-unimgr*. Aktualnie wspiera 3 rodzaje urządzeń sieciowych – *cisco-xr*, *cisco-xe* oraz *ovs*, które reprezentowane są jako osobne podmoduły, które trzeba zainstalować przed użyciem. W celu stworzenia usługi trzeba wskazać interfejsy końcowe (UNIs), między którym zostanie stworzone wirtualne połączenie (EVC).

a) OpenFlow

Od wersji Carbon, wirtualne połączenia tworzone są za pomocą filtrowania ruchu poprzez mechanizm dopasowywania pakietów (OpenFlow) po tagu VLAN. Na listingu 12 znajduje się wynik wywołania komendy *ovs-ofctl -O OpenFlow13 dump-flows s5* ukazujący tablicę reguł OpenFlow po stworzeniu usługi EVP-Line (*Ethernet Virtual Private Line*) pomiędzy

portami *s1-eth1* i *s5-eth1*, ze zdefiniowanym CE-Tag (VLANID klienta) równym 200.

```
1 root@mininet-vm:~/mgr# ovs-ofctl -O OpenFlow13 dump-flows s5
2 OFPST_FLOW reply (OF1.3) (xid=0x2):
3  cookie=0x0, duration=114.557s, table=0, n_packets=0, n_bytes=0,
priority=20,in_port=2,dl_vlan=200 actions=output:1
4  cookie=0x0, duration=114.552s, table=0, n_packets=0, n_bytes=0,
priority=20,in_port=3,dl_vlan=200 actions=output:1
5  cookie=0x0, duration=114.549s, table=0, n_packets=0, n_bytes=0,
priority=20,in_port=1,dl_vlan=200 actions=output:2,output:3
6  cookie=0x0, duration=114.569s, table=0, n_packets=23,
n_bytes=1955, priority=10,in_port=2 actions=output:3
7  cookie=0x0, duration=114.565s, table=0, n_packets=23,
n_bytes=1955, priority=10,in_port=3 actions=output:2
8  cookie=0x0, duration=114.561s, table=0, n_packets=0, n_bytes=0,
priority=0 actions=drop
```

Listing 12: Tablica reguł OpenFlow [źródło własne].

Wynik wywołania komendy zaprezentowany powyżej zawiera reguły przepływu na wirtualnym przełączniku *s5*. Numeracja portów jest analogiczna do ich nazw, tzn. port o numerze 1, to w nomenklaturze OVSDb *s5-eth1*. Jak widać, dostęp do portu 1 umożliwiają tylko reguły z linii 3 i 4, które oprócz portu, nakładają na wchodzący pakiet dodatkowy warunek posiadania tagu VLAN o identyfikatorze równym 200. W linii 5 przedstawiona jest obsługa ruchu wchodzącego do usługi, co znaczy, że wszystkie pakiety pojawiające się na porcie 1 zostają przekierowywane na porty 2 i 3 jedynie, gdy spełnią warunek zbieżności identyfikatora VLAN. W kolejnych dwóch liniach zdefiniowane są reguły zezwalające na ruch w sieci wewnętrznej mininet. Pakiety dopasowywane do ostatniej reguły (takie, które nie spełniły innych warunków) zostają odrzucone (linia 8).

b) OVSDb

Przed wydaniem najnowszej wersji ODL (Carbon), EVC było tworzone za pomocą tuneli GRE pomiędzy punktami końcowymi usługi, czyli za pomocą protokołu OVSDb. Fragment kodu modułu *odl-unimgr*, zaimplementowany w języku Java reprezentujący sposób tworzenia tunelu GRE za pośrednictwem OVSDb znajduje się poniżej.

```
1 /**
2  * Creates a submit a GRE tunnel to the Configuration DataStore.
```

```

3 * @param dataBroker An instance of the Data Broker to create a
transaction
4 * @param source The source UNI
5 * @param destination The destination UNI
6 * @param bridgeNode The bridge Node
7 * @param bridgeName The bridge name (example br0)
8 * @param portName The Port Name (example: eth0)
9 */
10 public static void createGreTunnel(DataBroker dataBroker,
11     Uni source,
12     Uni destination,
13     Node bridgeNode,
14     String bridgeName,
15     String portName) {
16     final InstanceIdentifier<TerminationPoint> tpId =
17         UnimgrMapper.getTerminationPointId(bridgeNode,
18             portName);
19     final OvsdbTerminationPointAugmentationBuilder
tpAugmentationBuilder =
20         new OvsdbTerminationPointAugmentationBuilder();
21     tpAugmentationBuilder.setName(portName);
22     final ArrayList<Options> options = Lists.newArrayList();
23     final OptionsKey optionKey = new OptionsKey("remote_ip");
24     final Options destinationIp = new OptionsBuilder()
25
.setOption(destination.getIpAddress().getIpv4Address().getValue())
26
.setKey(optionKey).setValue(destination.getIpAddress().getIpv4Addr
ess().getValue())
27     .build();
28     options.add(destinationIp);
29     tpAugmentationBuilder.setOptions(options);
30
tpAugmentationBuilder.setInterfaceType(SouthboundConstants.OVSDB_I
NTERFACE_TYPE_MAP.get("gre"));
31     // {irrelevant code ommited}
32     final TerminationPointBuilder tpBuilder = new
TerminationPointBuilder();
33     tpBuilder.setKey(InstanceIdentifier.keyOf(tpId));
34
tpBuilder.addAugmentation(OvsdbTerminationPointAugmentation.class,
tpAugmentationBuilder.build());
35     final WriteTransaction transaction =
dataBroker.newWriteOnlyTransaction();
36     transaction.put(LogicalDatastoreType.CONFIGURATION,
37         tpId,
38         tpBuilder.build());
39     transaction.submit();
40 }

```

Listing 13: Utworzenie tunelu GRE [15].

Na powyższym fragmencie kodu można zauważyć, że tunel GRE jest tworzony od portu źródłowego *portName* znajdującego się na wirtualnym przełączniku *bridgeName*. Najpierw tworzona jest instancja tzw. augmentacji portu (linia 19), po czym ustawiane są opcje portu (linia 29) i definiowany jest typ interfejsu, jako GRE (linia 30). W dalszej części kodu następuje

zapisanie do bazy danych *config*, co później dzięki modułowi *odl-ovsdb-southbound* jest przekazywane do rzeczywistej bazy danych urządzenia OVS.

5.3.3.3. Definicja QoS

a) OpenFlow

Jak to było przedstawione w podrozdziale 4.2.3 w punkcie a, opisującym konfigurację QoS za pomocą OpenFlow na podstawie dokumentacji, protokół ten posiada ciekawą koncepcję definicji jakości usług. Niestety, programowalny przełącznik OVS nie ma jeszcze zaimplementowanego wsparcia dla tworzenia *meter-table* [16], a co za tym idzie, nie można skonfigurować QoS za pomocą protokołu OpenFlow na urządzeniach OVS (niezależnie od wybranego kontrolera).

b) OVSDB

Definicję jakości w ODL poprzez OVSDB wykonuje się analogicznie do innych akcji wykonywanych na bazie danych OVS, czyli zapis docelowej konfiguracji w bazie danych *config* kontrolera, który przekazuje to do bazy danych przełącznika OVS. Definicja QoS w tym przypadku ogranicza się do ograniczenia prędkości przepływu, czyli ustawienia maksymalnej ilości kilobitów na sekundę przez dany port. Aby to zdefiniować, najpierw trzeba zdefiniować kolejkę (*queue*), która zawiera wspomniane ograniczenie, którą później przypisuje się do definicji QoS. Przykład dodania takiej kolejki za pomocą metody PUT na adresie URL *http://<controller-ip>:8181/restconf/config/network-topology:network-topology/topology/topology/ovsdb:1/node/ovsdb:HOST1/ovsdb:queues/QUEUE-1/* znajduje się na poniższym listingu.

```
1 {
2   "ovsdb:queues": [
3     {
4       "queue-id": "QUEUE-1",
5       "dscp": 25,
6       "queues-other-config": [
7         {
8           "queue-other-config-key": "max-rate",
9           "queue-other-config-value": "3600000"
```

```

10         }
11     ]
12 }
13 ]
14 }

```

Listing 13: Utworzenie kolejki QoS [14].

Na powyższym przykładzie tworzona jest kolejka o identyfikatorze *QUEUE-1*, ograniczająca szybkość przepływu do 3 600 000 kb/s.

5.3.4. Podsumowanie kontrolera

Kontroler ODL jest jednym z najbardziej znanych wolno dostępnych kontrolerów SDN. Jest też z pewnością w czołówce rozwijanych kontrolerów przez firmy związane domeną sieci komputerowych. W celu zwiększenia rozpoznawalności i udoskonalenia tego kontrolera, tworzone są międzynarodowe konferencje o nazwie *OpenDayLight Summit*, które świadczą o jego popularności.

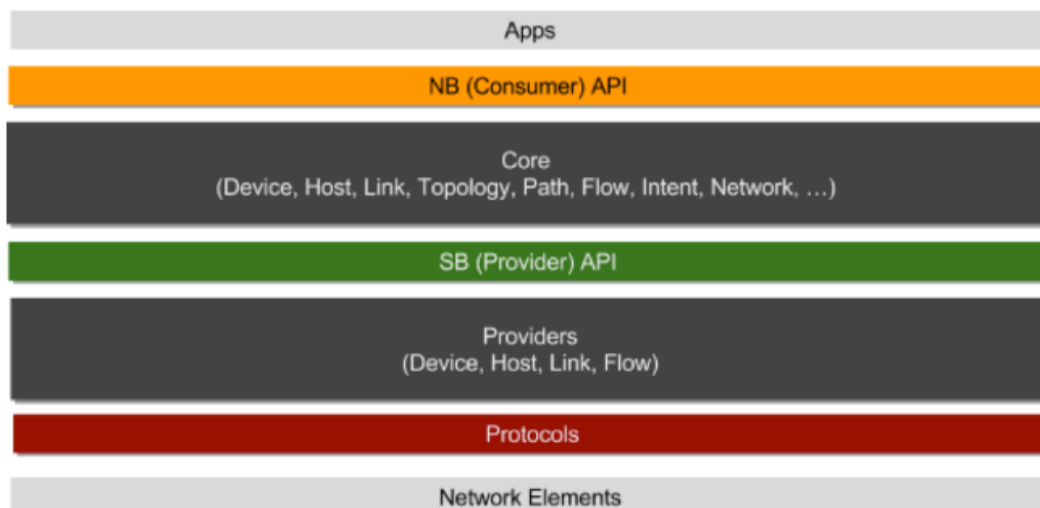
W wyznaczonych w niniejszej pracy kryteriach porównania protokołów OpenFlow i OVSDb, ODL pokazuje, że większym ograniczeniem, co do ich działania jest implementacja przełącznika OVS niż samego kontrolera. Dzięki modułom odpowiedzialnym za konfigurację urządzeń OVS za pomocą omawianych protokołów można wykonać praktycznie każdą możliwą akcję. Dodatkowo kontroler ODL posiada moduły zapewniające wyższy poziom abstrakcji, więc użytkownik nie musi wiedzieć, w jaki sposób jest to robione.

5.4. ONOS

5.4.1. Opis

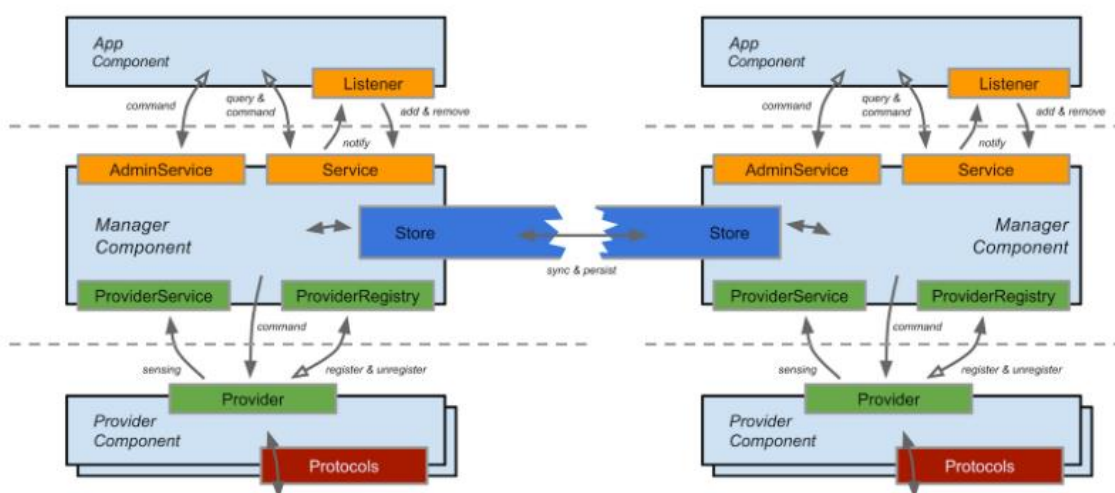
Następnym kontrolerem SDN wybranym do porównania protokołów OpenFlow i OVSDb jest ONOS. Jego pełna nazwa to *Open Network Operating System*. Oprogramowanie to zostało napisane w języku Java i podobnie jak OpenDayLight jego instancje są uruchamiane na kontenerze OSGI – karaf. Dzięki temu, kontroler ten może być uruchomiony na każdym systemie operacyjnym wspierającym JVM (*Java Virtual Machine*). Celem tego kontrolera jest stworzenie rozproszonego systemu, gdzie każda instancja może znajdować się w na innym miejscu [17].

Architekturę opisywanego kontrolera można przedstawić jako wiele współpracujących ze sobą poziomów reprezentujących jakąś funkcjonalność.



Rysunek 22: Warstwy kontrolera ONOS [17]

Jedną z najciekawszych warstw jest warstwa usługi (*Core*). Każdy taka usługa stanowi podsystem całego kontrolera. ONOS definiuje usługi odpowiadające za urządzenia, maszyny klienckie, połączenia między nimi, reguły przepływu i wiele innych poprzez zarządzanie ich bazą danych. W celu rozjaśnienia terminu podsystemu, na poniższym rysunku przedstawione są relacje pomiędzy jego komponentami [17].



Rysunek 23: Komponenty podsystemu ONOS [17]

Najniższą warstwą stosu kontrolera ONOS jest komponent o nazwie *Provider*, który jest odpowiedzialny za połączenie z urządzeniami sieciowymi za pomocą danych protokołów. W każdym podsystemie może istnieć jeden lub więcej *Provider*. Komponentem odpowiadającym za logikę usługi jest *Manager*, który otrzymuje informacje od dostawcy (*Provider*) i dostarcza je do innych podsystemów lub aplikacji.

Manager przesyła informacje do *Store*, którego zadaniem jest indeksowanie, zapisywanie i synchronizacja otrzymanych danych. Komponent ten odpowiada także za synchronizację instancji ONOSa pomiędzy sobą. Najwyżej usytuowanym w hierarchii komponentem jest komponent aplikacji (*Apps*), który konsumuje i manipuluje informacjami zebranymi przez *Manager'a*. Aplikacje posiadają szeroki zakres funkcjonalności, zaczynając od wyświetlania topologii po wyznaczania ścieżek w sieci [17].

ONOS zapewnia model reprezentujący sieć komputerową jako niezależny od protokołów graf skierowany. Interfejsy i ich implementacje znajdują się w module *org.onosproject.net*. Elementy sieci w modelu tego kontrolera to:

- *Device* – element infrastruktury sieciowej np. switch, ruter, access-point. Posiadają one zbiór interfejsów/portów oraz identyfikator urządzenia (*DeviceId*). Urządzenia są wewnętrznymi wierzchołkami grafu sieciowego,
- *Port* – Interfejs sieciowy na urządzeniu. Wraz z identyfikatorem urządzenia tworzą tzw. *ConnectPoint* reprezentujący punkt końcowy na krawędzi grafu,
- *Host* – Stacja końcowa sieci posiadająca adres IP, adres MAC, VLAN ID, oraz *ConnectPoint*. Jest to zewnętrzny wierzchołek grafu,
- *Link* – Skierowane połączenie pomiędzy urządzeniami. Reprezentuje wewnętrzną krawędź grafu,
- *EdgeLink* – Połączenie między urządzeniem a stacją końcową sieci, są to zewnętrzne krawędzie grafu,
- *Path* – Lista sąsiadujących połączeń (*Link* lub *EdgeLink*),
- *Topology* – Graf reprezentujący sieć komputerową. Aby obliczyć najlepszą ścieżkę pomiędzy punktami można użyć takich algorytmów jak BFS, Dijkstra lub Bellman-Ford [17].

Na poziomie aplikacji, dyrektywy dla sieci są wyrażane jako reguły przepływu na wysokim poziomie, podane jako pary kryteriów (*Match*) i działanie (*Action*). Instancja ONOS będzie odgrywać rolę dla dowolnego urządzenia, które albo zezwala na to, albo nie dopuszcza wprowadzanie zmian w urządzeniu [17].

ONOS jest projektem stworzonym przez ON.Lab (*Open Networking Lab*) składającym się głównie z AT&T i NTT Communications w Październiku 2015r na licencji Apache 2.0. Nazwy kolejnych wersji tego oprogramowania pochodzą od nazw ptaków w celu podkreślenia otwartego charakteru tego projektu (Możliwość

kontrybucji i korzystania przez każdego). Najnowsza wersja (1.10.0) wyszła w czerwcu 2017 roku i jest reprezentowana przez Zimorodka (*Kingfisher*) [17].

5.4.2. Uruchomienie kontrolera i połączenie z siecią mininet

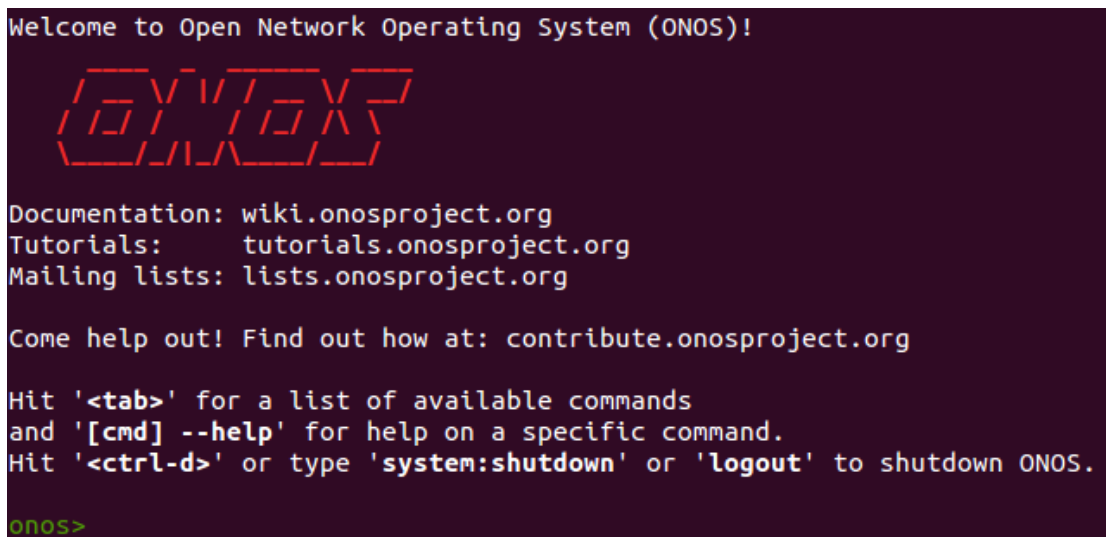
W celu uruchomienia ONOSa na lokalny komputer został ściągnięty gotowy obraz systemu operacyjnego Linux z zainstalowanym i skonfigurowanym kontrolerem, możliwy do znalezienia na stronie producenta [17]. Na przygotowanym obrazie wersja ONOS została zaktualizowana do najnowszej (1.10.0). Instancja programu została uruchomiona jako serwis, po czym uzyskany został dostęp do linii poleceń kontrolera za pomocą dostarczanego skryptu `~/onos/bin/onos.sh`.

W przygotowanej maszynie wirtualnej zainstalowane było oprogramowanie mininet. Na potrzeby pracy, topologia ta została zamieniona na tą przygotowaną w tej pracy. Tym razem mininet został wywołany za pomocą poniższej komendy.

```
#~/mininet/custom/mn --custom mgr.py --topo mgrtopo --controller  
remote,127.0.0.1 --mac --switch ovsk,protocols=OpenFlow13
```

Listing 14: Uruchomienie danej topologii mininet [źródło własne].

System uruchamia się i działa w podobny sposób jak opisywany wcześniej OpenDayLight, dlatego że działają na tym samym kontenerze OSGI. Poniżej znajduje się ekran uruchomionego już kontrolera.



Rysunek 24: Ekran startowy kontrolera ONOS [źródło własne]

Po uruchomieniu programu mininet z przygotowaną topologią, uruchamia się linia poleceń programu. Na początku wirtualne komputery nie mogą się ze sobą komunikować, co znaczy, że trzeba zainstalować w ONOS'ie odpowiedni moduł

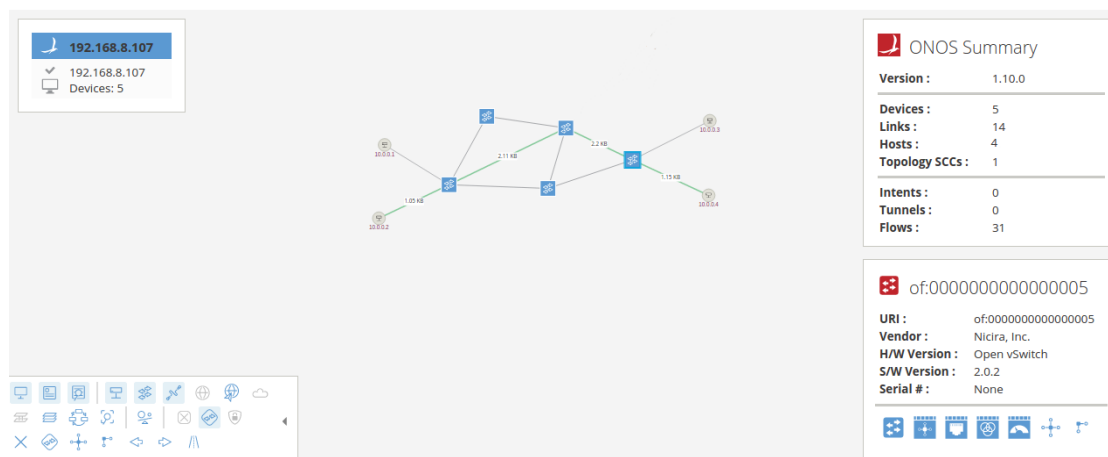
inicjalizujący reguły przepływu. W celu sprawdzenia aktywnych modułów w powłoce kontrolera można je wypisać za pomocą komendy `apps -a`, gdzie flaga specyfikuje, że interesują nas tylko aplikacje działające.

```
onos> apps -a -s
* 4  org.onosproject.ovsdbhostprovider  1.10.0  OVSDB host Provider
* 14 org.onosproject.optical-model      1.10.0  Optical information model
* 41 org.onosproject.lldpprovider       1.10.0  LLDP Link Provider
* 49 org.onosproject.drivers            1.10.0  Default device drivers
* 53 org.onosproject.ovsdb-base         1.10.0  OVSDB Provider
* 61 org.onosproject.hostprovider       1.10.0  Host Location Provider
* 62 org.onosproject.openflow-base     1.10.0  OpenFlow Provider
* 63 org.onosproject.openflow          1.10.0  OpenFlow Meta App
* 70 org.onosproject.ovsdb             1.10.0  OVSDB Southbound Meta App
```

Rysunek 25: Wypisanie aktywnych aplikacji w kontrolerze ONOS [źródło własne]

Na początku domyślnie zainstalowana jest tylko interfejs do urządzeń korzystających z protokołu OpenFlow, lecz niestety nie ma aplikacji odpowiedzialnej za inicjalizację reguł przepływu w podłączonych przełącznikach. W celu jej zainstalowania trzeba wpisać `app activate org.onosproject.fwd`. Po uaktywnieniu tej aplikacji możliwa jest komunikacja przełączników.


Przydatną cechą kontrolera ONOS jest jego interfejs graficzny, który jest dostępny na stronie <http://127.0.0.1:8181/onos/ui/index.html#/topo> (Rysunek 27).



Rysunek 26: Interfejs graficzny kontrolera ONOS [źródło własne]

Interfejs graficzny kontrolera ONOS zapewnia więcej funkcji niż interfejs opisywanego wcześniej ODL. W prawym górnym rogu można znaleźć podsumowanie mówiące o tym ile w topologii jest przełączników, ile stacji roboczych, ile połączeń, tuneli, a nawet reguł przepływu. Ponadto na zilustrowanej topologii można zobaczyć generowany ruch między wirtualnymi komputerami h2 (10.0.0.2) i h4 (10.0.0.4), który jest przedstawiony za pomocą zielonej linii.

Oprócz przedstawionych już funkcjonalności interfejsu graficznego, dodatkowo po kliknięciu na dane urządzenie lub połączenie można uzyskać informacje o nim. Po kliknięciu na przełącznik można otrzymać jego identyfikator, wersję oprogramowania, wersję protokołu OpenFlow i inne (Prawy dolny róg rysunku 27). Przy każdym wirtualnym switch'u znajdują się też przyciski przekierowujące na stronę ze statystykami z portów lub na stronę przedstawiającą reguły OpenFlow na danym urządzeniu (Rysunek 28).

Flows for Device of:0000000000000005 (5 total) 

Flow ID	App ID	Group ID	Table ID	Priority	Timeout	Permanent	State	Packets	Bytes
281477163465012	1	0	0	5	0	true	Added	4	392
Criteria: ETH_TYPE{ethType=800} Treatment Instructions: OUTPUT{port=CONTROLLER}									
281477163470778	1	0	0	5	0	true	Added	0	0
Criteria: ETH_TYPE{ethType=806} Treatment Instructions: OUTPUT{port=CONTROLLER}									
281477163470778	1	0	0	40000	0	true	Added	12	504
Criteria: ETH_TYPE{ethType=806} Treatment Instructions: OUTPUT{port=CONTROLLER}									
281477195151104	1	0	0	40000	0	true	Added	2768	224208
Criteria: ETH_TYPE{ethType=88cc} Treatment Instructions: OUTPUT{port=CONTROLLER}									
281477195264502	1	0	0	40000	0	true	Added	2768	224208
Criteria: ETH_TYPE{ethType=8942} Treatment Instructions: OUTPUT{port=CONTROLLER}									

Rysunek 27: Reguły OpenFlow w GUI kontrolera ONOS [źródło własne]

Korzystanie z aplikacji kontrolera ONOS jest możliwe na kilka sposobów. Pierwszy z nich to tworzenie plików konfiguracyjnych z rozszerzeniem json przedstawiający np. pożądaną konfigurację jednego z połączonych urządzeń i zapisanie ich w odpowiednim folderze (\$ONOS/tools/package/config). Tak zapisane pliki są wykonywane przez daną aplikację w momencie uruchamiania instancji kontrolera. Tak sformułowaną konfigurację (za pomocą pliku json) można w łatwiejszy sposób zaaplikować za pomocą REST API. Pomocnym w tym zadaniu jest Swagger przedstawiający dokumentację tego interfejsu dostępna na stronie <http://127.0.0.1:8181/onos/v1/docs/#/>, który udostępnia opcję wykonywania metod REST bezpośrednio z tej strony [17].



Rysunek 28: Uzyskanie informacji o topologii za pomocą metody GET [źródło własne]

Kolejnym interfejsem pozwalającym na konfigurację urządzeń jest linia poleceń kontrolera. Przykładowo w celu wypisania wszystkich urządzeń należy użyć komendy *devices* (Rysunek 30).

```
onos> devices -s
id=of:0000000000000001, available=true, role=MASTER, type=SWITCH, driver=ovs
id=of:0000000000000002, available=true, role=MASTER, type=SWITCH, driver=ovs
id=of:0000000000000003, available=true, role=MASTER, type=SWITCH, driver=ovs
id=of:0000000000000004, available=true, role=MASTER, type=SWITCH, driver=ovs
id=of:0000000000000005, available=true, role=MASTER, type=SWITCH, driver=ovs
```

Rysunek 29: Wypisanie dostępnych urządzeń przy użyciu ONOS CLI [źródło własne]

5.4.3. Analiza OpenFlow i OVSDB w kontrolerze

ONOS tak jak ODL posiada moduły odpowiadające za połączenie z przełącznikami OVS za pomocą OpenFlow i OVSDB, ich nazwy to *org.onosproject.openflow* oraz *org.onosproject.ovsdb*. Aby konfigurować urządzenia na wyższym poziomie abstrakcji, trzeba zainstalować odpowiednią aplikację, np. *org.onosproject.fwd* do inicjalizacji reguł OpenFlow w przełącznikach.

Moduł odpowiedzialny za OVSDB w wiki kontrolera ONOS jest bardzo słabo udokumentowany (opisane jest wyłącznie połączenie kontrolera z każdym z przełączników OVS), więc akcje związane z tym protokołem będą tylko opisane, bez faktycznych działań. Obecnie stworzone jest API w języku Java do zarządzania urządzeniami OVSDB pozwalające na tworzenie tuneli, mostków i konfiguracje portów

i interfejsów. Niestety nie ma stworzonej aplikacji do korzystania ze wspomnianego API [19].

5.4.3.1. Tagowanie VLAN

Tagowanie VLAN poprzez ONOS jest możliwe wyłącznie za pośrednictwem protokołu OpenFlow. Podobnie jak w przypadku kontrolera ODL, jest to możliwe poprzez interfejs REST. Dodanie pojedynczej reguły przepływu możliwe jest za pomocą metody POST na adres <http://192.168.8.107:8181/onos/v1/flows/of%3A0000000000000003?appId=org.onosproject.net.intent>, z poniższym ciałem metody.

```
1 {
2   "tableId": 0,
3   "appId": "org.onosproject.net.intent",
4   "priority": 1100,
5   "timeout": 0,
6   "isPermanent": true,
7   "deviceId": "of:0000000000000003",
8   "treatment": {
9     "instructions": [
10      {
11        "type": "OUTPUT",
12        "port": "2"
13      }
14    ],
15    "deferred": []
16  },
17  "selector": {
18    "criteria": [
19      {
20        "type": "IN_PORT",
21        "port": 1
22      },
23      {
24        "type": "VLAN_VID",
25        "vlanId": 200
26      }
27    ]
28  }
29 }
```

Listing 15: Reguła OpenFlow jako Ciało metody POST [źródło własne].

W powyższym zapytaniu dodawana jest reguła w priorytecie 1100 do wirtualnego przełącznika s3 (linia 7). W liniach 18-27 definiowane są warunki dopasowania reguły, czyli port o numerze 1 oraz VLANID równy 200. Akcja wyjściowa, czyli przekierowanie pakietu na port o numerze 2, zawarta jest w liniach 9-14.

5.4.3.2. Przypisywanie pakietów do EVC

W celu zdefiniowania usługi MEF, w kontrolerze ONOS można się posłużyć aplikacją VPLS (*Virtual Private Lan Service*). Celem VPLS jest łączenie wielu punktów końcowych w sieci OpenFlow, tworząc izolowane sieci na poziomie drugim warstwy OSI/ISO. Aktywuje się ją za pomocą polecenia *app activate org.onosproject.vpls* [17]. W celu stworzenia wirtualnej sieci między dwoma portami trzeba najpierw skonfigurować te porty, stworzyć VPLS, a następnie przypisać skonfigurowane interfejsy do tej sieci.

W niniejszej pracy zestawione zostanie połączenie między stacjami *h1* i *h3*. Poniższy rysunek przedstawia wykonanie tego zadania za pomocą linii poleceń kontrolera ONOS.

```
onos> vpls create mgrVpls
onos> interface-add -v 100 of:0000000000000001/1 h1
Interface added
onos> interface-add -v 100 of:0000000000000005/1 h3
Interface added
onos> vpls add-if mgrVpls h1
onos> vpls add-if mgrVpls h3
onos> vpls set-encap mgrVpls VLAN
```

Rysunek 30: Stworzenie sieci VPLS [źródło własne]

Na rysunku 31 przedstawione jest stworzenie usługi vpls o nazwie *mgrVpls*, po czym do dwóch krańcowych przełączników zostaje dodany interfejs VLAN o identyfikatorze równym 100. Następnie stworzone interfejsy zostają dodane do stworzonej usługi, po czym jest ona enkapsulowana za pomocą VLAN.

Po stworzeniu usługi w powyżej zaprezentowany sposób, do wirtualnych przełączników zostają dodane reguły OpenFlow pozwalające na ruch tylko pakietów posiadających tag VLAN o identyfikatorze 100 (Rysunek 15).

```
1 cookie=0x6e0000c6b5e494, duration=38.765s, table=0, n_packets=0,
n_bytes=0, send_flow_rem
priority=1100,in_port=1,d1_vlan=100,d1_dst=ff:ff:ff:ff:ff:ff
actions=output:4
2 cookie=0x6e0000cd23b85f, duration=38.751s, table=0, n_packets=0,
n_bytes=0, send_flow_rem
priority=1100,in_port=4,d1_vlan=100,d1_dst=ff:ff:ff:ff:ff:ff
actions=output:1
```

Listing 16: Dodane reguły OpenFlow przez moduł vpls kontrolera ONOS [źródło własne].

Podobnie jak w kontrolerze ODL, przypisywanie pakietów do EVC odbywa się za pomocą dodania reguł OpenFlow. Istnieje aplikacja do tworzenia tuneli za pomocą OVSDB, lecz niestety na razie istnieje tylko w wersji testowej i nie posiada jeszcze działającej implementacji.

5.4.3.3. Definicja QoS

Obecnie nie ma możliwości zdefiniowania jakości usługi poprzez kontroler ONOS na urządzeniach OVS. Można wysłać zapytania REST z konfiguracją *meter table*, lecz niestety tak jak to było wspomniane w ODL, programowalne przełączniki OVS nie posiadają implementacji tego mechanizmu. Nie można także zdefiniować prędkości przepływu za pomocą OVSDB, ponieważ ONOS nie dostarcza wystarczającego wsparcia dla tego protokołu.

5.4.4. Podsumowanie kontrolera

Kontroler ONOS podobnie jak ODL skupia wokół siebie duże zainteresowanie wielu firm związanych z domeną sieci komputerowych. W rezultacie porównania protokołów dowiedziałem się, że ten kontroler posiada znikome wsparcie dla OVSDB. Sprawa się ma zupełnie inaczej z protokołem OpenFlow, który jest zaimplementowany w dużo większym zakresie. Interfejs graficzny pozwala na wgląd zarówno na całą topologię sieci wirtualnej jak i na konkretne elementy wirtualnego przełącznika, dzięki czemu można w łatwy i szybki sposób dowiedzieć się jak wygląda i jak jest skonfigurowana podłączona sieć.

6. Porównanie wyników oraz wnioski

W niniejszym rozdziale zestawione zostaną wyniki porównania protokołów OpenFlow i OVSDB najpierw na podstawie dokumentacji a następnie w rzeczywistych przypadkach użycia. Analiza zostanie przeprowadzona w tabelach w celu lepszego uwidocznienia różnic.

6.1. Podsumowanie zestawienia protokołów na podstawie dokumentacji

Kryterium porównania	OpenFlow	OVSDB
Tagowanie VLAN	<ul style="list-style-type: none">- Zrealizowane za pomocą akcji,- Możliwe wszystkie operacje (dodawanie, edycja i usuwanie),- Możliwość dodawania więcej niż jednego taga.	<ul style="list-style-type: none">- Możliwość tworzenia różnych rodzajów portów, zezwalających na filtrowanie ruchu na różne sposoby,- Dodawanie pola VLAN do nieotagowanego pakietu możliwe tylko w określonych scenariuszach.
Przypisywanie pakietów do EVC	<ul style="list-style-type: none">- Przekierowywanie pakietów na daną ścieżkę możliwe przy pomocy zarówno tagów VLAN jak i bazując na portach.	<ul style="list-style-type: none">- Możliwość stworzenia tunelu do miejsca docelowego.
Definicja QoS	<ul style="list-style-type: none">- Możliwe ograniczanie prędkości przepływu.	<ul style="list-style-type: none">- Możliwość ograniczenia prędkości na danych portach.

Tabela 1: Teoretyczne porównanie protokołów [źródło własne]

Z porównania wynika, że oba protokoły potrafią wykonywać wszystkie operacje zdefiniowane przez kryteria. Trzeba jednak zauważyć, że przypisywanie pakietów do EVC w wykonaniu protokołu OVSDB nie jest dobrym rozwiązaniem i nie pokrywa większości scenariuszy, jakie mogłyby być zdefiniowane przez dostawcę usługi. W przypadku kryterium Tagowania VLAN również protokół OpenFlow oferuje większe możliwości, gdyż nie ma ograniczeń, co do konkretnych rozwiązań typu porty *access* czy *trunk*. W przypadku ostatniego kryterium w teorii oba protokoły wypadają w zestawieniu podobnie, gdyż oba potrafią ograniczyć ruch, tylko za pomocą innych sposobów. Podsumowując, oba protokoły mają duże możliwości, ale w tym konkretnym ich zestawieniu to OpenFlow przedstawia lepsze perspektywy.

6.2.Podsumowanie zestawienia protokołów w rzeczywistych przypadkach użycia

6.2.1. Kontroler OpenDayLight

Kontroler ten porozumiewa się z przełącznikami OVS za pomocą opisywanych protokołów przy użyciu modułów *odl-openflowplugin* oraz *odl-ovsdb*. Po zapisaniu konfiguracji w odpowiednim miejscu w bazie danych, wspomniane moduły są odpowiedzialne za przeniesienie konfiguracji na urządzenia OVS. Interakcje z bazą danych kontrolera są możliwe za pomocą protokołu RESTCONF.

Kryterium porównania	OpenFlow	OVSDB
Tagowanie VLAN	- Możliwość dodawania, edycji i usuwania akcji OpenFlow odpowiedzialnych za manipulacje tagiem VLAN.	- Możliwość tworzenia różnych rodzajów portów (access, trunk), zezwalających na filtrowanie ruchu na różne sposoby,
Przypisywanie pakietów do EVC	- Aplikacja <i>odl-unimgr</i> zapewniająca możliwość tworzenia usług MEF, a co za tym idzie, wirtualnych połączeń EVC (za pomocą filtrowania po tagu VLAN regułami OpenFlow).	- Możliwość stworzenia tunelu GRE między portami.
Definicja QoS	- Zaimplementowane wsparcie do manipulowania <i>meter table</i> , odpowiedzialne za ograniczanie prędkości przepływu (Niestety brak implementacji w OVS).	- Możliwość stworzenia kolejek ograniczających prędkości na danych portach.

Tabela 2: Porównanie protokołów w kontrolerze ODL [źródło własne]

Kontroler ODL zapewnia duże wsparcie zarówno dla protokołu OpenFlow jak i dla OVSDB. Co prawda nie można zdefiniować jakości usługi za pomocą protokołu OpenFlow, jednakże nadrabia elastycznością w innych kryteriach porównania.

6.2.2. Kontroler ONOS

Aplikacje pozwalające na komunikację kontrolera ONOS z przełącznikami OVS to *org.onosproject.openflow* oraz *org.onosproject.ovsdb*, które działają na takiej samej zasadzie jak moduły w ODL. Niestety moduł OVSDB jest udokumentowany w minimalnym zakresie i najprawdopodobniej nie da się skonfigurować urządzeń za pomocą OVSDB w tym kontrolerze.

Kryterium porównania	OpenFlow	OVSDB
Tagowanie VLAN	- Możliwość dodawania, edycji i usuwania akcji OpenFlow odpowiedzialnych za manipulacje tagiem VLAN.	-
Przypisywanie pakietów do EVC	- Aplikacja <i>org.onosproject.vpls</i> zapewniająca możliwość tworzenia wirtualnych usług, (tworzenie wirtualnego połączenia za pomocą filtrowania ruchu po tagu VLAN regułami OpenFlow).	-
Definicja QoS	- Zaimplementowane wsparcie do manipulowania <i>meter table</i> , odpowiedzialne za ograniczanie prędkości przepływu (Niestety brak implementacji w OVS).	-

Rysunek 31: Porównanie protokołów w kontrolerze ODL [źródło własne]

Kontroler ONOS pozwala na wykonanie niemalże takich samych akcji jak kontroler ODL w zakresie protokołu OpenFlow, jednakże robi to w nieco inny sposób. W przypadku OVSDB istnieje działające Java API zezwalające na konfigurację pewnych elementów, jednakże najprawdopodobniej nie zostało one jeszcze zaimplementowane w konkretnej aplikacji.

7. Podsumowanie

W mojej pracy starałem się porównać dwa protokoły komunikacyjne OpenFlow i OVSDB w sieciach programowalnych SDN, gdzie jako kryterium porównania obrałem elementy składowe usług zgodnych ze standardem Carrier Ethernet 2.0. W celu wykonania starannego zestawienia protokołów najpierw porównane zostały na podstawie ich dokumentacji oraz innych źródeł, a następnie w rzeczywistych przypadkach użycia.

W pierwszym etapie wykonana została szczegółowa analiza specyfikacji obu protokołów i dodatkowych źródeł. Z tego porównania wynika, że oba protokoły potrafią wykonywać wszystkie operacje zdefiniowane przez kryteria, każdy na swój sposób. Mimo tego, że za pomocą protokołu OVSDB można wykonać wszystkie zdefiniowane zadania, niektóre z nich może wykonać w ściśle określonych przypadkach.

W następnym etapie (praktycznym) za pomocą oprogramowania mininet stworzona została wirtualna sieć komputerowa składająca się z przełączników OVS. W celu lepszej analizy możliwości protokołów zostały one porównane w dwóch kontrolerach SDN – OpenDayLight i ONOS.

W pierwszym kontrolerze komunikacja z wirtualnymi przełącznikami okazała się być zaimplementowana w bardzo wysokim stopniu umożliwiając wykonanie prawie wszystkich akcji zdefiniowanych przez kryteria porównania. Podczas wykonywania zadań można było zauważyć większą predyspozycję jednego protokołu nad drugim w danym zadaniu. Do przypisywania pakietów do danego połączenia EVC lepiej nadaje się OpenFlow, jako że ten protokół służy właśnie do kierowania ruchem w sieci. Natomiast w przypadku definicji QoS lepiej spisał się protokół OVSDB, jednakże gdyby konfigurowane były urządzenia z zaimplementowanym mechanizmem do definicji jakości usługi za pomocą OpenFlow, wynik byłby podobny.

Drugi kontroler umożliwia wykonanie tych samych akcji OpenFlow, co ODL w nieco inny sposób. Dodatkowo umożliwia zapoznanie się z topologią i poszczególnymi jej komponentami za pomocą przejrzystego GUI. Obsługa protokołu OVSDB natomiast wymaga lepszej implementacji i dodatkowej dokumentacji.

Podsumowując, porównywane protokoły działają w zupełnie inny sposób i służą innym celom, lecz mogą być wykorzystywane przy wykonywaniu tych samych zadań. W przypadku zdefiniowanych w tej pracy kryteriów porównania OpenFlow wydaje się być bardziej uniwersalnym protokołem i często lepiej implementowanym w kontrolerach SDN niż OVSDB.

Bibliografia

- [1] Paul Goransson, Chuck Black. Software Defined Networks: A Comprehensive Approach. Elsevier, 2014.
- [2] Open vSwitch. Getting Started. <http://docs.openvswitch.org/en/latest/intro>, 2016. [dostęp: 2017-05-20].
- [3] Rajdeep Dua. Open vSwitch deep dive. <https://www.slideshare.net/rajdeep/openvswitch-deep-dive>, 2013. [dostęp: 2017-05-20].
- [4] D. Marschke, J. Doyle, P. Moyer. SDN: Anatomy of OpenFlow Volume I. Lulu, 2015.
- [5] SdxCentral. What is Open vSwitch Database or OVSDb? <https://www.sdxcentral.com/cloud/open-source/definitions/what-is-ovsdb/>, [dostęp: 2017-05-21]
- [6] B. Davie, The Open vSwitch Database Management Protocol, <https://tools.ietf.org/html/draft-pfaff-ovsdb-proto-04>, 2013, [dostęp: 2017-05-21]
- [7] J. Kieffer, R. Satitiro, MEF-CECP Study Guide for Carrier Ethernet Professionals, Fujitsu Network Communications Inc., 2015.
- [8] Open Networking Foundation, OpenFlow Switch Specification, <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf>, 2014, [dostęp: 2017-05-23]
- [9] Open vSwitch, Open vSwitch Database Schema, <http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf>, 2015, [dostęp: 2017-05-24]
- [10] Sdx Central, What is Open Virtual Network (OVN)? How it works, <https://www.sdxcentral.com/sdn/network-virtualization/definitions/what-is-open-virtual-network-ovn-how-it-works/>, [dostęp: 2017-06-07]
- [11] Wikipedia, OpenDaylight Project, https://en.wikipedia.org/wiki/OpenDaylight_Project, [dostęp: 2017-06-10]
- [12] OpenDayLight, Wiki OpenDaylight, <https://wiki.opendaylight.org>, [dostęp: 2017-06-10]
- [13] Wikipedia, YANG, <https://en.wikipedia.org/wiki/YANG>, [dostęp: 2017-06-13]
- [14] OpenDayLight, Docs OpenDayLight, <http://docs.opendaylight.org>, [dostęp: 2017-06-14]
- [15] OpenDayLight, Gerrit OpenDayLight, <https://git.opendaylight.org/gerrit/#/q/project:unimgr>, [dostęp: 2017-06-15]
- [16] Open vSwitch, Mail Open vSwitch, <https://mail.openvswitch.org/pipermail/ovs-discuss/2016-April/040754.html>, 2016, [dostęp: 2017-06-15]
- [17] ONOS, Wiki ONOS, <https://wiki.onosproject.org>, [dostęp: 2017-06-16]
- [18] Cloudify, Open Source, End-to-End MEF CE 2.0 Service Provisioning with NFV Overlay PoC From Amartus, Cisco, and Cloudify, <http://cloudify.co/2016/12/08/end-to-end-provisioning-standardized-mef-eline-service-multi-vendor-sdn-environment.html>, 2016, [dostęp: 2017-06-16]
- [19] Google Groups, Onos Project, <https://groups.google.com/a/onosproject.org/forum/#!topic/onos-dev/D6-ywuUyv14>, 2016, [dostęp: 2017-06-18]