Warsztat 3

mgr inż. Jan Palimąka

Zadanie (15 pkt)

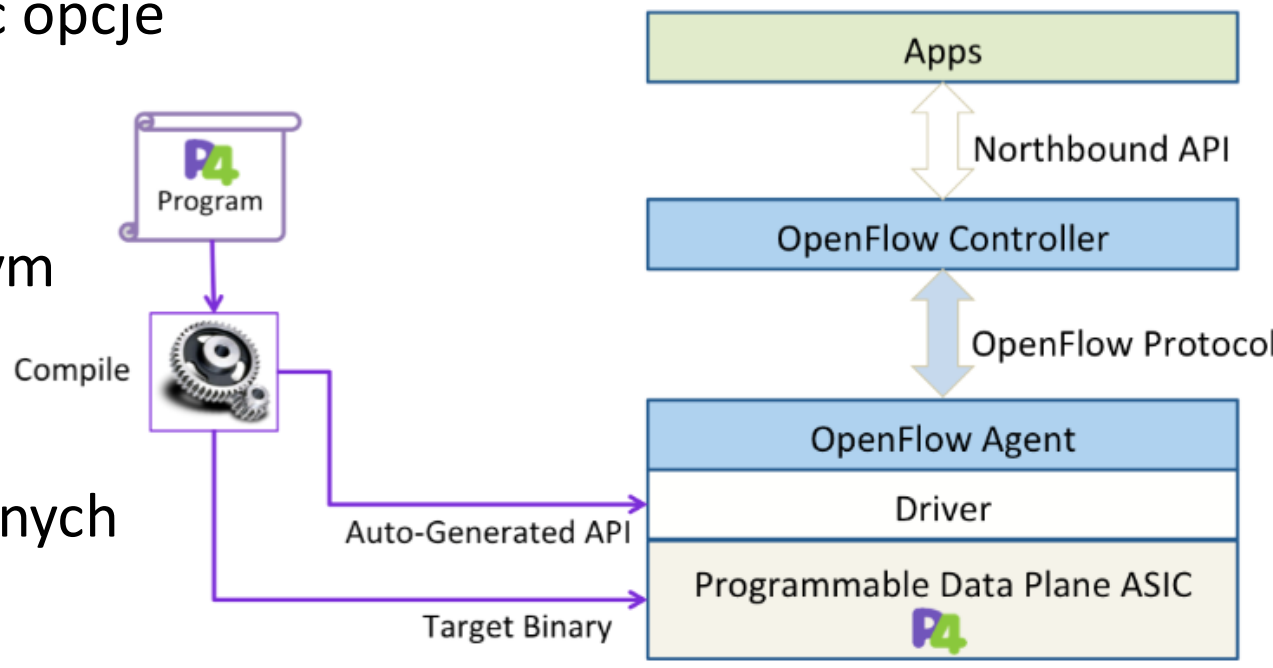
- Napisanie programu P4 (P4_16), który może pełnić funkcje:
  - Przekazywania pakietów między interfejsami
  - Dodawania lub usuwania tagu VLAN
- Uruchomienie switcha bmv2 z napisanym programem P4
- Przetestowanie działania
  - `ping`
  - `scapy`

Sprawozdanie

- Zamieszczony kod źródłowy
- Wpisywane polecenia (wraz z krótkim opisem), które:
  - Nie były wymienione w instrukcji
  - Należało "odkryć" wg instrukcji
- Termin
  - Grupa poniedziałkowa: 25 marca (do końca dnia)
  - Grupa środowa: 27 marca (do końca dnia)

Praca z programem P4

- Napisanie (lub zmiana) programu P4
  - Dowolny edytor tekstu
- Skompilowanie programu - dla bmv2 powstanie plik w formacie JSON (jakie powinny być opcje dla kompilatora?)
  - `p4c`
  - `p4c-bm2-ss`
- Uruchomienie switcha ze skompilowanym programem
  - `mininet`
  - `simple_switch` (jakie opcje?)
- Skonfigurowanie wpisów w tabelach i innych obiektach (jak?)
  - `simple_switch_CLI`



Mininet i BMv2

- Potrzebny plik: https://github.com/p4lang/behavioral-model/blob/main/mininet/p4_mininet.py
- Przykład: https://github.com/p4lang/behavioral-model/blob/main/mininet/1sw_demo.py
- Uruchomienie przykładu: `./1sw_demo.py [opcje]`

```
from p4_mininet import P4Switch, P4Host
...

class P4SwitchTopo(Topo):
    ...
    def __init__(self, ...):
        ...
        switch = self.addSwitch("nazwa",
            sw_path = "/usr/local/bin/simple_switch",
            json_path = "...",
            ...)
    ...

def main():
    ...
    net = Mininet(topo = ...,
        host = P4Host,
        switch = P4Switch,
        controller = None,
        ...)
```

Konfiguracja wpisów w tabeli – simple_switch_CLI

https://github.com/p4lang/behavioral-model/blob/main/docs/runtime_CLI.md

- table_add
- table_set_default
- table_delete

v1model – szkielet programu P4

```
#include <core.p4>
#include <v1model.p4>

header h1_t {
    bit<N> f1;
    ...
}

header h2_t {
    ...
}

struct headers {
    h1_t h1;
    h2_t h2;
}

struct metadata {
}
```

Typy danych:

- `bit<N>`
- `int`
- `error`
- `typedef` *`typ`* `nazwa_typu`

Metody nagłówków (`header`):

- `setValid()`
- `setInvalid()`
- `isValid()`

v1model – szkielet programu P4

```
parser parserI(packet_in pkt, // zdefiniowane w core.p4
               out headers hdr, // zdefiniowane na poprzednim slajdzie
               inout metadata meta, // jak wyżej
               inout standard_metadata_t stdmeta) // zdefiniowane w v1model.p4
{
    state start {
        pkt.extract(hdr.h1);
        transition ...; // nazwa kolejnego stanu, accept, reject albo select()
    }
}

control verifyChecksum(inout headers hdr,
                       inout metadata meta)
{
    apply {}
}
```

v1model – szkielet programu P4

```
control cIngress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t stdmeta)
{
    /* definicje tabel, zmiennych, itp. */
    apply {
        /* wykonywane operacje na nagłówkach i/lub pakiecie */
    }
}

control cEgress(inout headers hdr,
                inout metadata meta,
                inout standard_metadata_t stdmeta)
{
    /* definicje tabel, zmiennych, itp. */
    apply {
        /* wykonywane operacje na nagłówkach i/lub pakiecie */
    }
}
```

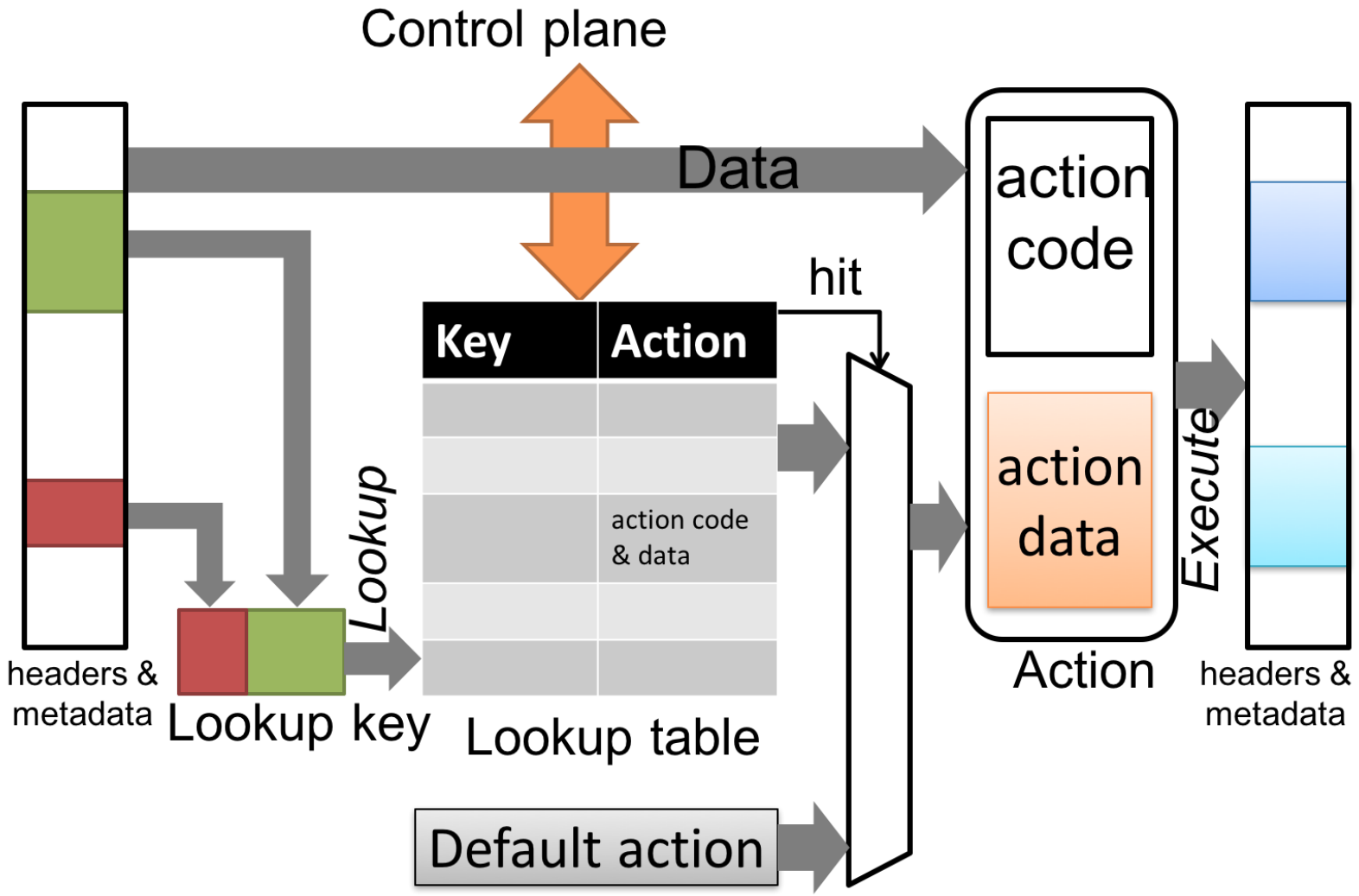
v1model – szkielet programu P4

```
control updateChecksum(inout headers hdr,
                       inout metadata meta)
{
    apply {}
}

control DeparserI(packet_out packet,
                  in headers hdr)
{
    apply {
        packet.emit(hdr.h1);
        ...
    }
}
```

```
V1Switch(
    parserI(),
    verifyChecksum(),
    cIngress(),
    cEgress(),
    updateChecksum(),
    DeparserI()
) main;
```

Tabele



Źródło: https://staging.p4.org/p4-spec/docs/P4-16-v1.2.4.html

Tabela – składnia

```
table nazwa_tabeli {
    // obowiązkowe
    key = { expr1; expr2; ... }
    action = { action1; action2; ... }
    // opcjonalne
    [const] default_action = action3(parametry);
    size = N; // maksymalna liczba wpisów w tabeli
    const entries = { ... } // wpisy statyczne (niemodyfikowalne),
        // nie można dodać nowych wpisów po kompilacji
    direct_counter = dc; // przypisanie instancji dc typu direct_counter do tabeli
    direct_meter = dm; // przypisanie instancji dm typu direct_meter do tabeli
    implementation = ...; // zmiana sposobu działania tabeli,
        // tylko z action_profile lub action_selector
}
```

Wykonanie tabeli (w bloku apply): `nazwa_tabeli.apply()`

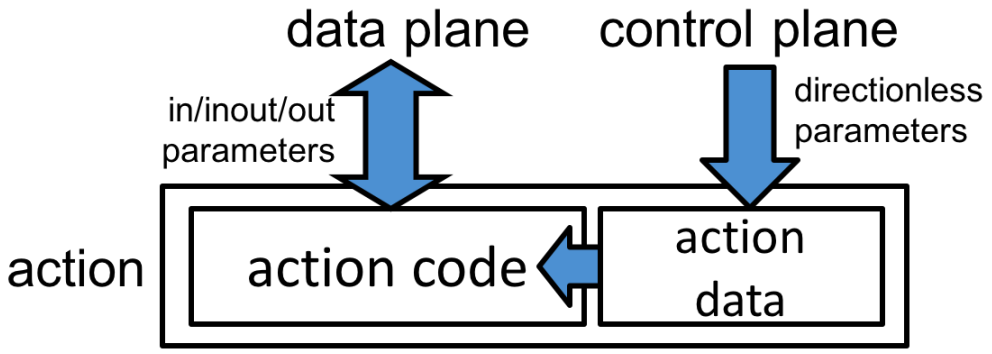
Tabela – klucz

```
key = {
    expression1 : match_kind annotation;
    expression2 : match_kind annotation;
    ... }
```

- expressionN - dowolne wyrażenie (praktycznie wszystko, co zwraca wartość)
- match_kind - rodzaj dopasowania
  - exact - dokładne dopasowanie
  - ternary - maska bitowa
  - lpm - Longest Prefix Match
  - range - zakres od do
  - selector - potrzebne dla action_selector, nie jest częścią klucza
  - optional
- match_kind - rodzaj dopasowania
- annotation - opcjonalne adnotacje
  - @name("inna_nazwa")

Tabela – akcja

- Reprezentuje wykonanie jakiejś akcji lub działania
- Jedna akcja może wywoływać inne akcje
- Nie powinny zawierać instrukcji warunkowych
- Wartości parametrów mogą być zdefiniowane w płaszczyźnie sterowania
- Może korzystać ze wszelkich obiektów (zmiennych) w zakresie, w którym jest zdefiniowana akcja
- Nie zwraca wartości



Źródło: https://staging.p4.org/p4-spec/docs/P4-16-v1.2.4.html

Przydatne materiały

- P4_16 Language Specification - https://staging.p4.org/p4-spec/docs/P4-16-v1.2.4.html
- Opis modelu v1model - https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4
- Podstawowa biblioteka P4: https://github.com/p4lang/p4c/blob/main/p4include/core.p4
- Przykładowe programy dla bmv2 (wszystkie kończące się na "-bmv2.p4"): https://github.com/p4lang/p4c/tree/main/testdata/p4_16_samples
- Kompilator P4C - https://github.com/p4lang/p4c
- Switch bmv2 - https://github.com/p4lang/behavioral-model
- Tutoriale P4:
  - https://github.com/p4lang/tutorials
  - https://github.com/opennetworkinglab/ngsdn-tutorial