Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Pokročilé síťové technologie II

Projekt

Implementace adaptivního firewallu pomocí jazyka P4

Vypracovali: Boháčová Jana, Keberle Ondřej, Mlýnek Jakub, Otava Michal, Doubravský Ondřej

Datum: 29.11.2024
Forma studia: prezenční
ak. r.: 2023/2024

Obsah

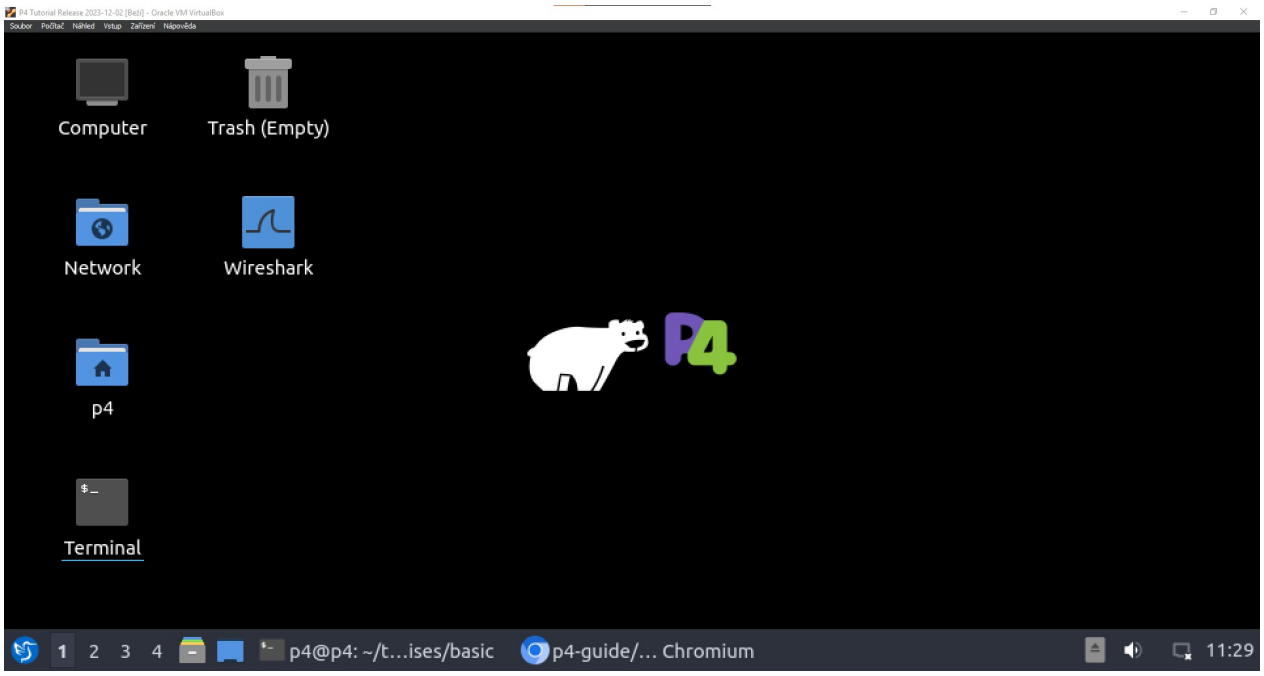
1 Instalace prostředí

Způsobů instalace prostředí by se dalo nalézt několik, my jsme však zvolili zřejmě nejjednodušší a nejpraktičtější cestu, kdy jsme si stáhli „VM image“ s předinstalovanými nástroji pro jazyk P4.

V našem případě bylo prvním krokem tedy stáhnutí „Release VM image“ z 2.12.2023, který byl již otestován i na OS Windows. Po dokončení stáhování stačilo otevřít soubor ve Virtual Boxu a došlo k instalaci prostředí s OS Lubuntu 20.04.

Po instalaci už prostředí spustíme a přihláíme se pomocí připraveného uživatelského profilu „p4“ a hesla „p4“.

Po příhlášení už jsme přímo v předpřipraveném prostředí s už nainstalovanými nástroji pro jazyk P4. [1]



Obrázek 1: Plocha připraveného prostředí

1.1 Předpřipravené ukázky

V předpřipraveném prostředí je k dispozici několik již implementovaných kódu, které mají za cíl demonstrovat jazyk P4. Jednotlivé ukázky jsou ve složce „/home/p4/tutorials/excersises“. Jednotlivé ukázky jsou dále roztřízeny do složek, ve kterých se nachází:

- readme.md soubor
- soubor(y) definující topologii sítě pro mininet
- nevyřešený P4 kód
- vyřešený P4 kód
- Makefile

Jednotlivé ukázky se spouští příkazem „make run“, který provede následující:

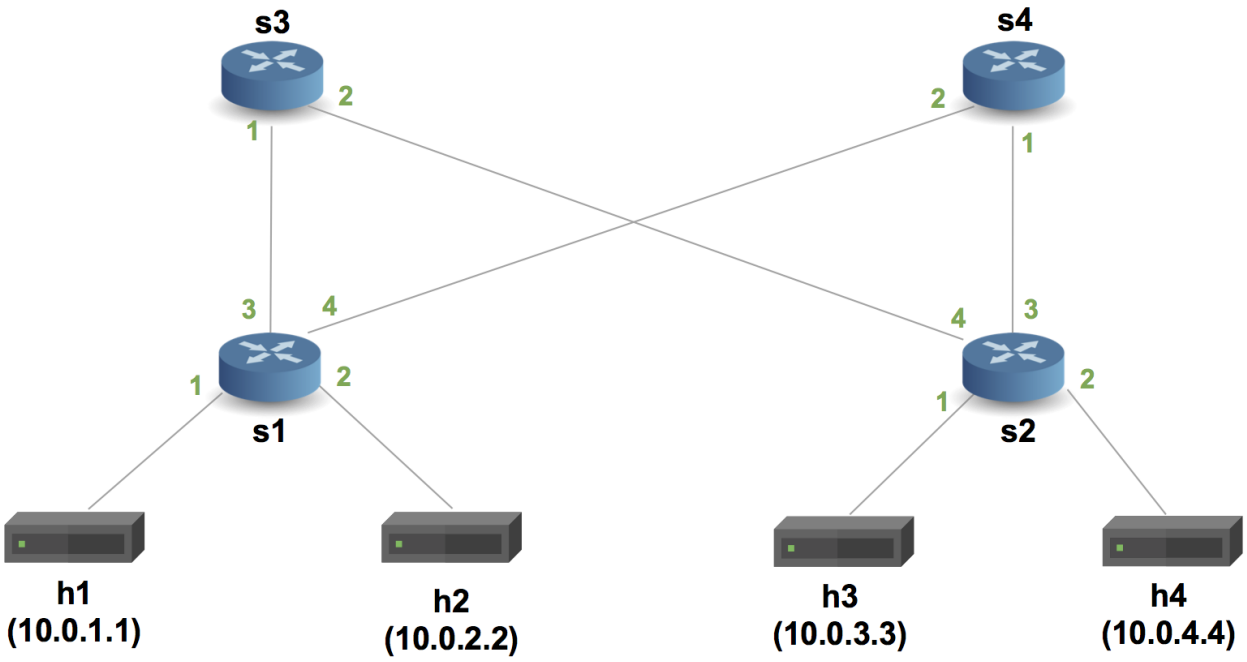
- přeloží P4 kód (do JSON souborů)
- spustí mininet s předdefinovanou topologií, nahraje konfigurací jednotlivých switchů dle vzniklých JSON souborů
- nakonfiguruje všechny hosty (IP adresy, defaultní cesty, ARP záznamy)
- spustí logování jednotlivých switchů do souborů ve složce „logs“
- spustí tcpdump na všech rozhraních jednotliých switchů

Pro vypnutí se používá příkaz „make stop“. Příkazem „make clean“ poté smažeme vzniklé dočasné soubory (pcap, logy, ...).

2 Popis struktury základního P4 programu - basic.p4

Pro implementaci adaptivního firewallu jsme zvolili základní ukázku - „basic.p4“, kterou budeme později rozšiřovat o nové funkcionality. V této části dokumentu se nachází popis zmíněného příkladu.

Na obrázku 2 je zobrazena topologie příkladu. Topologie je složena ze čtyř switchů a čtyř hostů.



Obrázek 2: Topologie příkladu basic.p4

2.1 Headers

Header je část paketu (umístěna na začátku), ve které lze nalézt informace o využitém protokolu, IP adresách atd.

V P4 lze definovat typy zahláví, určovat v nich obsažená pole a jejich velikost. Hlavičky můžeme vytvářet sami nebo lze využít již předdefinovaných hlaviček, které se v sítích běžně využívají, jako například Etherner header, IP header, TCP header, UDP header a další.

V programu „basic.p4“ lze nalézt header ethernet_t a ipv4_t. Header ipv4_t obsahuje například pole s verzí, TTL, protokolem nebo zdrojovou a cílovou IP adresou a header ethernet_t obsahuje pole ethertype a dále pole pro zdrojovou a cílovou MAC adresu.

2.2 Parser

Parsery dokáží analyzovat pakety, identifikovat přítomnost záhlaví v nich obsažených a vybrat příslušné pole na základě definovaných typů těchto záhlaví.

V programu „basic.p4“ je funkce MyParser, která využívá již zmíněné ethernet_t a ipv4_t headery. Na zakládě těchto hlaviček zpracovává pakety. V přichozím paketu je identifikována ethernetová hlavička a zkontrolováno pole EtherType. Pokud je typ tohoto pole TYPE_IPV4, tak dále analyzuje IPv4 hlavičku. Pokud ne, tak je paket přijat bez dalšího zpracování.

2.3 Checksum verification

Slouží k ověření kontrolního součtu hlaviček. V programu „basic.p4“ je funkce pouze definována, ale uvnitř nejsou žádne další instrukce.

2.4 Ingress processing

Zde se řeší zpracování příchozích paketů. V našem případě směrování IP paketu na základě cílové adresy („action ipv4_forward(macAddr_t dstAddr, egressSpec_t port)“) pomocí směrovací tabulky („table ipv4_lpm...“). Pokud není nalezena shoda, paket se zahodí („action = drop()“).

2.5 Egress processing

V této části programu se řeší zpracování odchozích paketů. V našem případě neřeší.

2.6 Checksum computation

Tato část programu je určena k výpočtu kontrolního součtu pro IP záhlaví. Kontroluje se správnost IP adres, délky, TTL atd. („hdr.ipv4.isValid(), hdr.ipv4.version,...“).

2.7 Deparser

Deparser je poslední část kódu. Sestavuje jednotlivé záhlaví zpracovaného paketu zpět do podoby, ve které mohou být odeslány do sítě.

[1, 2, 3]

3 Implementace adaptivního firewallu

Hlavním cílem tohoto projektu, je vytvořit adaptivní firewall, který bude reagovat na ICMP packety, tj. na základě množství ICMP packetů za určitý časový interval, bude povolovat nebo blokovat provoz.

3.1 Základní počítání IPv4 packetů

3.1.1 Úprava kódu

Prvním krokem bylo vyzkoušet, jakým způsobem P4 umožňuje počítat packety. K této funkcionalitě slouží v P4 konstrukce zvaná **counter**. Pro prvotní testování byla použita základní topologie, která byla vytvořena v příkladu „basic.p4“. P4 kód v této ukázce byl doplněn o countery počítající průchozí IPv4 packety skrz jednotlivé porty.

V sekci „Ingress processing“ programu jsme definovali nový counter nazvaný „port_counter“. Tento counter je typu „packets_and_bytes“, počítá tedy jak packety tak byty.

```
86  control MyIngress(inout headers hdr,
87                    inout metadata meta,
88                    inout standard_metadata_t standard_metadata) {
89
90      counter(512, CounterType.packets_and_bytes) port_counter;
91
```

Obrázek 3: Definice nového counteru

Níže v kódu jsme v akci „ipv4_forward“ přidali řádek s metodou „.count“, která zajišťovala samotné počítání packetů a bytů. Argumentem této metody je index (číslo) counteru, do kterého chceme hodnoty přičítat. V tomto případě je argumentem číslo portu, kterým packety prošli (standard_metadata.ingress_port). Tento řádek tedy počítá jednotlivé packety a byty na jednotlivých portech a ukládá tyto hodnoty do counterů indexovaných podle čísel portů (čísla portů viz. obrázek 2). [4, 2, 5, 6]

```
96      action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
97          standard_metadata.egress_spec = port;
98          hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
99          hdr.ethernet.dstAddr = dstAddr;
100         hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
101         port_counter.count((bit<32>)standard_metadata.ingress_port);
102     }
```

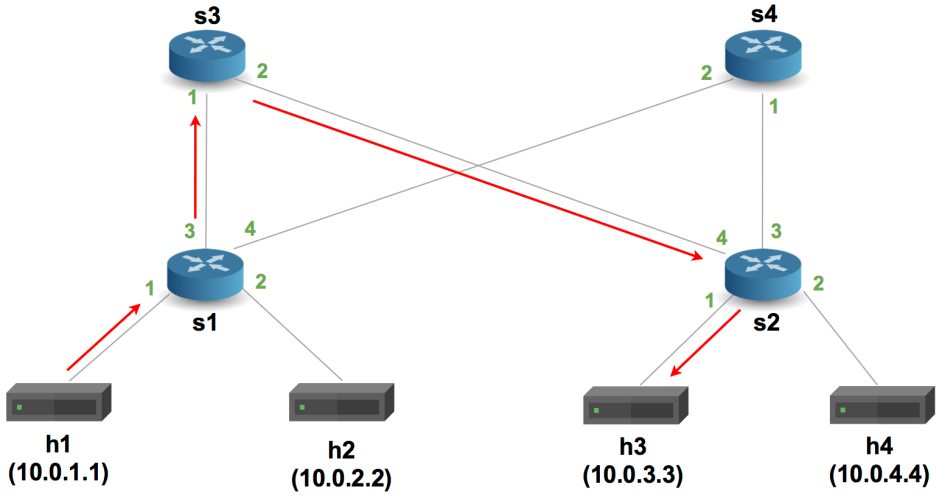
Obrázek 4: Definice akce pro inkrementaci counteru

3.1.2 Čtení hodnot z counteru

Čtení hodnot z counteru lze nejjednodušším způsobem provádět pomocí tzv. „Runtime CLI“, což je utilita která umožňuje základní ovládání jednotlivých switchů pomocí příkazů.

Každý switch v topologii dostane přidělen tzv. thrift port, pomocí kterého se k jednotlivým switchům připojíme. V testované topologie se jedná o porty 9090 - 9093 pro switche s1 - s4. Při otestování

konektivity mezi hostem h1 a h3 příkazem ping, budou ICMP packety procházet sítí tak, jak je vyznačeno v obrázku 5. Očekáváme tedy že se budou zvyšovat hodnoty counterů 1 a 3 na switchi s1, 1 a 2 na switchi s3 a 4 a 1 na switchi s2. Jednotlivé hodnoty counterů jsou zobrazeny na obr. 6. [7, 5, 4]



Obrázek 5: Cesta packetů mezi h1 a h3

```
p4@p4:~/tutorials/exercises/counter$ simple_switch_CLI --thrift-port 9090
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: counter_read MyIngress.port_counter 1
MyIngress.port_counter[1]= (980 bytes, 10 packets)
RuntimeCmd: counter_read MyIngress.port_counter 3
MyIngress.port_counter[3]= (980 bytes, 10 packets)
```

(a) s1

```
p4@p4:~/tutorials/exercises/counter$ simple_switch_CLI --thrift-port 9091
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: counter_read MyIngress.port_counter 4
MyIngress.port_counter[4]= (980 bytes, 10 packets)
RuntimeCmd: counter_read MyIngress.port_counter 1
MyIngress.port_counter[1]= (980 bytes, 10 packets)
```

(b) s2

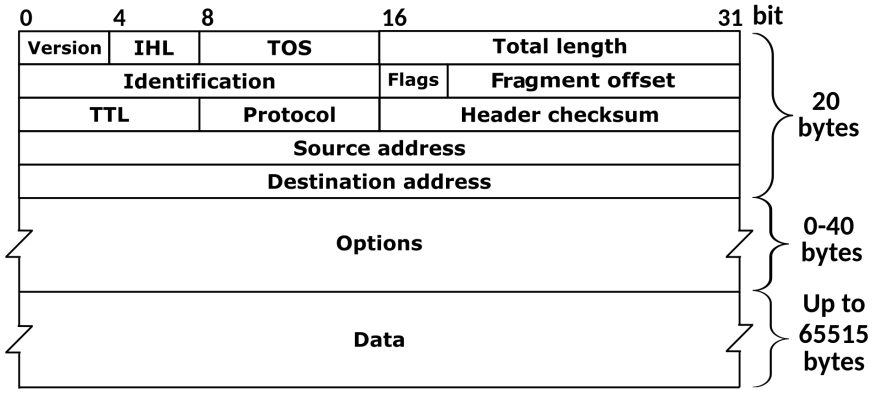
```
p4@p4:~/tutorials/exercises/counter$ simple_switch_CLI --thrift-port 9092
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: counter_read MyIngress.port_counter 1
MyIngress.port_counter[1]= (980 bytes, 10 packets)
RuntimeCmd: counter_read MyIngress.port_counter 2
MyIngress.port_counter[2]= (980 bytes, 10 packets)
```

(c) s3

Obrázek 6: Test counterů

3.2 Rozpoznávání ICMP packetů

Jelikož adaptivní firewall bude reagovat na ICMP packety, bylo další logickým krokem sestavit část programu, která bude odlišovat ICMP packety od ostatního provozu. Toto odlišení bylo provedeno na základě porovnávání hodnoty pole „protocol“ v IPv4 záhlaví (obr. 7), která je pro ICMP packety rovna 1 (viz. [8]). Do kódu bylo přidáno několik částí, které měli tuto klasifikaci provozu provádět. Jednalo se o **tabulku**, **akci** a **podmínku pro aplikování tabulky**.



Obrázek 7: IP záhlaví

```
142     table icmp_match {
143         key = {
144             hdr.ipv4.protocol: exact;
145         }
146         actions = {
147             icmp_process;
148             drop;
149             NoAction;
150         }
151         size = 1024;
152         default_action = drop();
153     }
```

Obrázek 8: Tabulka „icmp_match“

```
126     action icmp_process() { //akce pro zpracovani icmp
127         icmp_counter.count(1);
128     }
```

Obrázek 9: Akce „icmp_process“

```
155         apply {
156             if (hdr.ipv4.isValid()) {
157                 if(hdr.ipv4.protocol == ICMP) {
158                     icmp_match.apply();
159                 }
160                 ipv4_lpm.apply();
161 
162             }
163         }
164     }
```

Obrázek 10: Podmínka pro aplikování tabulky „icmp_match“

Po přidání tohoto kódu a otestování pingem ale bohužel klasifikace ICMP nebyla funknčí, viz. logy na switchi s1.

```
[07:38:58.660] [bmv2] [D] [thread 38607] [78.0] [cxt 0] Pipeline 'ingress': start
[07:38:58.660] [bmv2] [T] [thread 38607] [78.0] [cxt 0] basic.p4(156) Condition "hdr.ipv4.isValid()" (node_2) is true
[07:38:58.660] [bmv2] [T] [thread 38607] [78.0] [cxt 0] basic.p4(157) Condition "hdr.ipv4.protocol == ICMP" (node_3) is true
[07:38:58.660] [bmv2] [T] [thread 38607] [78.0] [cxt 0] Applying table 'MyIngress.icmp_match'
[07:38:58.660] [bmv2] [D] [thread 38607] [78.0] [cxt 0] Looking up key:
* hdr.ipv4.protocol   : 01

[07:38:58.660] [bmv2] [D] [thread 38607] [78.0] [cxt 0] Table 'MyIngress.icmp_match': miss
[07:38:58.660] [bmv2] [D] [thread 38607] [78.0] [cxt 0] Action entry is MyIngress.drop -
[07:38:58.660] [bmv2] [T] [thread 38607] [78.0] [cxt 0] Action MyIngress.drop
```

Obrázek 11: Nefunkční klasifikace ICMP

Ukázalo se, že toto je zapříčíněno způsobem, jakým jsou naplňovány tabulky jednotlivých switchů. Jelikož se jedná o základní ukázkový příklad, jsou jednotlivé záznamy definovány ručně, viz. níže, čímž. mj. odpadá nutnost řešit ARP.

```
"table_entries": [
  {
    "table": "MyIngress.ipv4_lpm",
    "default_action": true,
    "action_name": "MyIngress.drop",
    "action_params": { }
  },
  {
    "table": "MyIngress.ipv4_lpm",
    "match": {
      "hdr.ipv4.dstAddr": ["10.0.1.1", 32]
    },
    "action_name": "MyIngress.ipv4_forward",
    "action_params": {
      "dstAddr": "08:00:00:00:01:11",
      "port": 1
    }
  },
```

Obrázek 12: Příklad záznamů v tabulce

Pro korektní identifikaci ICMP provozu bylo tedy nutné do tabulek všech switchů (soubor sX-runtime.json) vložit následující záznam.

```
{
  "table": "MyIngress.icmp_match",
  "match": {
    "hdr.ipv4.protocol": [1]
  },
  "action_name": "MyIngress.icmp_process",
  "action_params": { }
}
```

Obrázek 13: Záznam v tabulce pro zpracování ICMP

Po tomto kroku se již akce definované v tabulce korektně spouštěly a tudíž počítání ICMP packetů bylo funknční, což dokazují obrázky níže.

3.2.1 Ověření funkčnosti

Na obrázku 14 je zobrazen výpis logů, který ukazuje, že byla správně vyhodnocena sekvence klasifikace ICMP:

(hdr.ipv4.protocol == ICMP) → (icmp_match) → (icmp_process) → (icmp_counter.count(...))

```
[09:08:48.095] [bmv2] [D] [thread 42651] [105.0] [cxt 0] Pipeline 'ingress': start
[09:08:48.095] [bmv2] [T] [thread 42651] [105.0] [cxt 0] basic.p4(156) Condition "hdr.ipv4.isValid()" (node_2) is true
[09:08:48.095] [bmv2] [T] [thread 42651] [105.0] [cxt 0] basic.p4(157) Condition "hdr.ipv4.protocol == ICMP" (node_3) is true
[09:08:48.095] [bmv2] [T] [thread 42651] [105.0] [cxt 0] Applying table 'MyIngress.icmp_match'
[09:08:48.095] [bmv2] [D] [thread 42651] [105.0] [cxt 0] Looking up key:
* hdr.ipv4.protocol   : 01

[09:08:48.095] [bmv2] [D] [thread 42651] [105.0] [cxt 0] Table 'MyIngress.icmp_match': hit with handle 0
[09:08:48.095] [bmv2] [D] [thread 42651] [105.0] [cxt 0] Dumping entry 0
Match key:
* hdr.ipv4.protocol   : EXACT     01
Action entry: MyIngress.icmp_process -

[09:08:48.095] [bmv2] [D] [thread 42651] [105.0] [cxt 0] Action entry is MyIngress.icmp_process -
[09:08:48.095] [bmv2] [T] [thread 42651] [105.0] [cxt 0] Action MyIngress.icmp_process
[09:08:48.095] [bmv2] [T] [thread 42651] [105.0] [cxt 0] basic.p4(126) Primitive (bit<32>)standard_metadata.ingress_port
[09:08:48.095] [bmv2] [T] [thread 42651] [105.0] [cxt 0] basic.p4(126) Primitive icmp_counter.count((bit<32>)standard_metadata.ingress_port)
[09:08:48.095] [bmv2] [T] [thread 42651] [105.0] [cxt 0] Updated counter 'MyIngress.icmp_counter' at index 3
```

Obrázek 14: Funkční klasifikace ICMP

Na obrázcích 15 a 16 je poté zobrazeno otestování funkčnosti counterů.

Na obrázku 15a) bylo provedeno 10 pingů, což odpovídá counterům na obrázku 15b). Na tomto obrázku vidíme 20 ICMP packetů, což odpovídá 10 pingům (10 · ECHO-REQUEST + 10 · ECHO-REPLY).

Na obrázku 16a) byl proveden test jiného provozu než ICMP (telnet), což se v counterech (obrázek 16b)) projevilo přičtením jednoho packetu do port_counteru, ale nikoliv do icmp_counteru. Počítání ICMP packetů tedy funguje správně.

```
mininet> h1 ping h3
PING 10.0.3.3 (10.0.3.3) 56(84) bytes of data.
64 bytes from 10.0.3.3: icmp_seq=1 ttl=61 time=10.7 ms
64 bytes from 10.0.3.3: icmp_seq=2 ttl=61 time=13.1 ms
64 bytes from 10.0.3.3: icmp_seq=3 ttl=61 time=8.91 ms
64 bytes from 10.0.3.3: icmp_seq=4 ttl=61 time=14.1 ms
64 bytes from 10.0.3.3: icmp_seq=5 ttl=61 time=10.1 ms
64 bytes from 10.0.3.3: icmp_seq=6 ttl=61 time=30.3 ms
64 bytes from 10.0.3.3: icmp_seq=7 ttl=61 time=11.2 ms
64 bytes from 10.0.3.3: icmp_seq=8 ttl=61 time=10.7 ms
64 bytes from 10.0.3.3: icmp_seq=9 ttl=61 time=9.90 ms
64 bytes from 10.0.3.3: icmp_seq=10 ttl=61 time=9.90 ms
^C
--- 10.0.3.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9036ms
rtt min/avg/max/mdev = 8.908/12.896/30.287/5.983 ms
```

(a) ping

```
p4@p4:~$ simple_switch_CLI --thrift-port 9090
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: counter_read MyIngress.icmp_counter 1
MyIngress.icmp_counter[1]= (1960 bytes, 20 packets)
RuntimeCmd: counter_read MyIngress.port_counter 1
MyIngress.port_counter[1]= (980 bytes, 10 packets)
```

(b) Hodnoty counterů

Obrázek 15: Test počítání ICMP packetů - ping

```
mininet> h1 telnet h3
Trying 10.0.3.3...
telnet: Unable to connect to remote host: Connection refused
```

(a) telnet

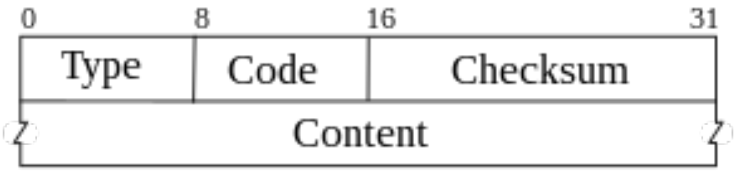
```
p4@p4:~$ simple_switch_CLI --thrift-port 9090
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: counter_read MyIngress.icmp_counter 1
MyIngress.icmp_counter[1]= (1960 bytes, 20 packets)
RuntimeCmd: counter_read MyIngress.port_counter 1
MyIngress.port_counter[1]= (1054 bytes, 11 packets)
```

(b) Hodnoty counterů

Obrázek 16: Test počítání ICMP packetů - telnet

3.2.2 Odlišení REQUEST a REPLY zpráv

Jelikož nechceme duplicitní hodnoty počtu pingů, bylo nutné odlišit REQUEST a REPLY zprávy. Toto odlišení bylo provedeno na základě porovnávání hodnoty „code“ uvnitř ICMP záhlaví (obr. 21), která je pro REQUEST zprávy = 8 a pro REPLY = 0 (viz. [9]).



Obrázek 17: ICMP záhlaví

Abychom se k tomuto poli dostali, potřebovali jsme parsovat ICMP záhlaví. Do P4 kódu bylo tedy dodáno: definice ICMP záhlaví, ICMP parser, podmínka pro odlišení REQUEST a REPLY zpráv, Deparser.

```
44  header icmp_t {
45      bit<8>    type;
46      bit<8>    code;
47      bit<16>   checksum;
48      bit<32>   unused;
49  }
50
51  struct headers {
52      ethernet_t   ethernet;
53      ipv4_t       ipv4;
54      icmp_t       icmp;
55  }
```

Obrázek 18: Definice ICMP záhlaví

```
78      state parse_ipv4 {
79          packet.extract(hdr.ipv4);
80          transition select(hdr.ipv4.protocol){
81              ICMP: parse_icmp;
82              default: accept;
83          }
84      }
85
86      state parse_icmp {
87          packet.extract(hdr.icmp);
88              transition accept;
89          }
90      }
```

Obrázek 19: ICMP parser

```
158     apply {
159         if (hdr.ipv4.isValid()) {
160             if(hdr.ipv4.protocol == ICMP) {
161                 if(hdr.icmp.type == ICMP_ECHO_REPLY){
162                     icmp_match.apply();
163                 }
164             }
165             ipv4_lpm.apply(); //routing
166 
167         }
168     }
169 }
```

Obrázek 20: Odlišení REQUEST a REPLY zpráv

```
207 control MyDeparser(packet_out packet, in headers hdr) {
208     apply {
209         packet.emit(hdr.ethernet);
210         packet.emit(hdr.ipv4);
211         packet.emit(hdr.icmp);
212     }
213 }
```

Obrázek 21: Deparser

V tomto stavu bylo funknčí počítání pouze ICMP ECHO-REPLY zpráv.

3.3 Implementace kontroleru

Dalším krokem bylo implementace kontroleru, který bude rozhodovat o povolení/zablokování provozu. Kostra Python kódu pro kontroler byla převzata z ukázkového kódu „p4runtime“, ze kterého jsme použili funkce „printCounter“, „readTableRules“, „printGrpcError“ a „main“, do kterého jsme později doplnili další funkce.

3.3.1 Komunikace kontroleru se switchem

Kontroler komunikuje s jednotlivými switchi pomocí tzv. „P4Runtime“. Zatímco P4 je jazyk pro programování „data plane“ jednotlivých síťových zařízení, P4Runtime je „control plane“ specifikace, která zajišťuje „doručení“ jednotlivých pravidel do switchů.

Rozhraní API P4Runtime je implementováno programem, který spouští server gRPC. Tento program se nazývá "P4Runtime server". Server musí ve výchozím nastavení naslouchat na portu TCP 9559, což je port, který byl pro službu P4Runtime přidělen organizací IANA.

3.3.2 Základní kostra kontroleru, vyčítání hodnot counterů

Základem kontroleru je metoda „main“. V této metodě provádíme v základní podobě následující akce:

- navázání spojení k jednotlivým switchům
- odeslání „master arbitration update“ zpráv
- vyčtení záznamů tabulek z jednotlivých switchů
- vyčítání požadovaných counterů co 2 sek.

Kód metody „main“ je zobrazen na obr. 23. Ukázkový výpis vyčítání counterů je zobrazen na obr. 22

```
----- Reading counters -----
s1 MyIngress.port_counter 1: 38 packets (3724 bytes)
s2 MyIngress.icmp_counter 1: 0 packets (0 bytes)
s3 MyIngress.icmp_counter 1: 0 packets (0 bytes)
s4 MyIngress.icmp_counter 1: 0 packets (0 bytes)

----- Reading counters -----
s1 MyIngress.port_counter 1: 40 packets (3920 bytes)
s2 MyIngress.icmp_counter 1: 0 packets (0 bytes)
s3 MyIngress.icmp_counter 1: 0 packets (0 bytes)
s4 MyIngress.icmp_counter 1: 0 packets (0 bytes)

----- Reading counters -----
s1 MyIngress.port_counter 1: 42 packets (4116 bytes)
s2 MyIngress.icmp_counter 1: 0 packets (0 bytes)
s3 MyIngress.icmp_counter 1: 0 packets (0 bytes)
s4 MyIngress.icmp_counter 1: 0 packets (0 bytes)
```

Obrázek 22: Vyčítání counterů pomocí kontroleru

```
60  def main(p4info_file_path, bmv2_file_path):
61      p4info_helper = p4runtime_lib.helper.P4InfoHelper(p4info_file_path)
62
63      try:
64          s1 = p4runtime_lib.bmv2.Bmv2SwitchConnection(
65              name='s1',
66              address='127.0.0.1:50051',
67              device_id=0,
68              proto_dump_file='logs/s1-p4runtime-requests.txt')
69          s2 = p4runtime_lib.bmv2.Bmv2SwitchConnection(
70              name='s2',
71              address='127.0.0.1:50052',
72              device_id=1,
73              proto_dump_file='logs/s2-p4runtime-requests.txt')
74          s3 = p4runtime_lib.bmv2.Bmv2SwitchConnection(
75              name='s3',
76              address='127.0.0.1:50053',
77              device_id=2,
78              proto_dump_file='logs/s3-p4runtime-requests.txt')
79          s4 = p4runtime_lib.bmv2.Bmv2SwitchConnection(
80              name='s4',
81              address='127.0.0.1:50054',
82              device_id=3,
83              proto_dump_file='logs/s4-p4runtime-requests.txt')
84
85          s1.MasterArbitrationUpdate()
86          s2.MasterArbitrationUpdate()
87          s3.MasterArbitrationUpdate()
88          s4.MasterArbitrationUpdate()
89
90          readTableRules(p4info_helper, s1)
91          readTableRules(p4info_helper, s2)
92          readTableRules(p4info_helper, s3)
93          readTableRules(p4info_helper, s4)
94
95          while True:
96              sleep(2)
97              print('\n----- Reading counters -----')
98              printCounter(p4info_helper, s1, "MyIngress.port_counter", 1)
99              printCounter(p4info_helper, s2, "MyIngress.icmp_counter", 1)
100             printCounter(p4info_helper, s3, "MyIngress.icmp_counter", 1)
101             printCounter(p4info_helper, s4, "MyIngress.icmp_counter", 1)
```

Obrázek 23: Metoda „main“

3.3.3 Funkcionalita pro manipulaci s tabulkama

Aby bylo možné povolovat nebo zakazovat provoz, bylo potřeba zajistit způsob, jakým bychom mohli přidávat nebo odebírat pravidla z tabulek.

Pro přidávání pravidla slouží funkce „writeIpForwardRule“, která na základě předaných parametrů přidá nové pravidlo do tabulky „ipv4_lpm“.

```
48  def writeIpForwardRule(p4info_helper, sw, dst_ip_addr, dst_mac_addr, port):
49      """
50      Instalace pravidla do tabulky IPv4 forward
51      """
52      table_entry = p4info_helper.buildTableEntry(
53          table_name="MyIngress.ipv4_lpm",
54          match_fields={
55              "hdr.ipv4.dstAddr": (dst_ip_addr, 32)
56          },
57          action_name="MyIngress.ipv4_forward",
58          action_params={
59              "dstAddr": dst_mac_addr,
60              "port": port
61          })
62      sw.WriteTableEntry(table_entry)
63      print("Installed IPv4 forward rule on %s" % sw.name)
```

Obrázek 24: writeIpForwardRule

Pro odebírání pravidel z tabulky slouží funkce „deleteIpForwardRule“, která na základě předaných parametrů odstraní pravidlo z tabulky „ipv4_lpm“. Tato funkce závisí na funkci switche „DeleteTableEntry“, která není ve switchi defaultně implementovaná, tudíž ji bylo nutné do souboru „utils/p4runtime_lib/switch.py“ dodat.

```
72  def deleteIpforwardRule(p4info_helper, sw, dst_ip_addr, dst_mac_addr, port):
73      """
74      Odstraneni pravidla z tabulky IPv4 forward
75      """
76      table_entry = p4info_helper.buildTableEntry(
77          table_name="MyIngress.ipv4_lpm",
78          match_fields={
79              "hdr.ipv4.dstAddr": (dst_ip_addr, 32)
80          },
81          action_name="MyIngress.ipv4_forward",
82          action_params={
83              "dstAddr": dst_mac_addr,
84              "port": port
85          })
86      sw.DeleteTableEntry(table_entry)
87      print("Removed IPv4 forward rule from %s" % sw.name)
```

Obrázek 25: deleteIpForwardRule

3.3.4 Implementace logiky blokování/povolování provozu

Logika adaptivního firewallu spočívá v počítání ICMP (ECHO-REPLY) packetů procházející switchi s1 a s2. V případě že na daném switchi překročí počet těchto packetů práhovou hodnotu (defaultně 10) za určitý čas (defaultně 10s), odstraní se z tabulky na daném switchi pravidla pro IP směrování na portech 1 a 2. Zároveň se nastaví proměnná „sX_blocked_flag“ na hodnotu True, abychom zamezili pokusům o mazání pravidel, které již na switchi neexistují.

Pokud hodnota průchozích packetů později klesne pod mez, pravidla pro směrování se opět nainstalují (čímž se opětovně povolí provoz) a proměnná „sX_blocked_flag“ se nastaví na False.

Toto rozhodování probíhá v nekonečné smyčce.

Tuto funkci ověříme např. příkazem ping s parametrem -i. Tento parametr nastavuje interval mezi jednolivými odeslanými ECHO-REQUEST zprávami. Například příkazem ping -i 0.5 vyšleme 2 packety za sekundu, čímž zablokujeme provoz. Po uplynutí 10s se provoz znovu povolí, za 10s znovu zablokuje, atd...

```
root@p4:/home/p4/tutorials/exercises/adaptivni_fw# python3 controller.py
Blokuju provoz na s1, porty 1,2; Pocet ICMP za poslednich 10 vterin::20, prahova hodnota je: 10
Removed IPv4 forward rule from s1
Removed IPv4 forward rule from s1
Povoluju provoz na s1, porty 1,2; Pocet ICMP za poslednich 10 vterin::0,  prahova hodnota je: 10
Installed IPv4 forward rule on s1
Installed IPv4 forward rule on s1
```

Obrázek 26: Ukázka funkce controlleru

```
s1_blocked_flag = False
s2_blocked_flag = False

while True:
    icmp_counter_difference_s1 = getCounterPacketCountDifference(p4info_helper, s1, "MyIngress.icmp_counter", 1)
    icmp_counter_difference_s2 = getCounterPacketCountDifference(p4info_helper, s2, "MyIngress.icmp_counter", 1)

    if (icmp_counter_difference_s1 > kriticka_hodnota_ICMP):
        if (not s1_blocked_flag):
            print(f"Blokuju provoz na s1, porty 1,2; Pocet ICMP za poslednich {prumerovaci_doba} vterin::{icmp_counter_difference_s1}, prahova hodnota je: {kriticka_hodnota_ICMP}")
            deleteIpforwardRule(p4info_helper, sw=s1, dst_ip_addr="10.0.1.1", dst_mac_addr="08:00:00:00:01:11", port=1)
            deleteIpforwardRule(p4info_helper, sw=s1, dst_ip_addr="10.0.2.2", dst_mac_addr="08:00:00:00:02:22", port=2)
            s1_blocked_flag = True
    else:
        if (s1_blocked_flag):
            print(f"Povoluju provoz na s1, porty 1,2; Pocet ICMP za poslednich {prumerovaci_doba} vterin::{icmp_counter_difference_s1},  prahova hodnota je: {kriticka_hodnota_ICMP}")
            writeIpForwardRule(p4info_helper, sw=s1, dst_ip_addr="10.0.1.1", dst_mac_addr="08:00:00:00:01:11", port=1)
            writeIpForwardRule(p4info_helper, sw=s1, dst_ip_addr="10.0.2.2", dst_mac_addr="08:00:00:00:02:22", port=2)
            s1_blocked_flag = False

    if (icmp_counter_difference_s2 > kriticka_hodnota_ICMP):
        if (not s2_blocked_flag):
            print(f"Blokuju provoz na s2, porty 1,2;  Pocet ICMP za poslednich {prumerovaci_doba} vterin::{icmp_counter_difference_s2}, prahova hodnota je: {kriticka_hodnota_ICMP}")
            deleteIpforwardRule(p4info_helper, sw=s2, dst_ip_addr="10.0.3.3", dst_mac_addr="08:00:00:00:03:33", port=1)
            deleteIpforwardRule(p4info_helper, sw=s2, dst_ip_addr="10.0.4.4", dst_mac_addr="08:00:00:00:04:44", port=2)
            s2_blocked_flag = True
    else:
        if (s2_blocked_flag):
            print(f"Povoluju provoz na s2, porty 1,2; Pocet ICMP za poslednich {prumerovaci_doba} vterin::{icmp_counter_difference_s1}, prahova hodnota je: {kriticka_hodnota_ICMP}")
            writeIpForwardRule(p4info_helper, sw=s2, dst_ip_addr="10.0.3.3", dst_mac_addr="08:00:00:00:03:33", port=1)
            writeIpForwardRule(p4info_helper, sw=s2, dst_ip_addr="10.0.4.4", dst_mac_addr="08:00:00:00:04:44", port=2)
            s2_blocked_flag = False
```

Obrázek 27: Blokování/povolování provozu

4 Závěr

Hlavním účelem semestrálního projektu bylo implementovat pomocí jazyku P4 adaptivní firewall. Pro hlubší pochopení dané problematiky byla nejprve popsáná instalace prostředí a lehké seznámění s předpřipravenými kódy, které mají za úkol uživatele seznámit s možnostmi jazyka P4. Jednomu z těchto kódů bylo věnováno o něco více času, neboť se jednalo o kód k topologii, ze které se vycházelo i v dalších částech projektu. Součástí kapitoly, která se tomuto kódu věnuje je tak popis jedotlivých částí kódů včetně jejich funkcionalit.

Nejrozsáhlejší část práce se zaobírala implementací adaptivního firewallu, který měl reagovat na ICMP packety a na základě jejich množství za časový interval povolovat nebo blokovat provoz. Prvním úspěšným krokem bylo počítání paketů funkcí „counter“ a vyčítání hodnot z counteru. Jelikož adaptivní firewall měl za úkol reagovat na ICMP packety, tak na základě hodnoty pole „protocol“ v IPv4 záhlaví jsme začali odlišovat jiné packety právě od ICMP packetů. Tento krok bylo pro jeho špatnou funkčnost nutno opravit správným způsobem naplňování tabulek.

Následující část práce se týkala ověřením funkčnosti, kde jsme ověřovali správnost vyhodnocení klasifikace ICMP. Byl proto proveden i test jiného provozu - telnet. Úspěšným výsledkem testu bylo přičtení packetu do port_counteru (ale nikoliv do icmp_counteru). Samotná adaptivnost firewallu pak spočívala v implementaci logiky, která řešila počítání ICMP (ECHO-REPLY) za určitou jednotku času. Dle počtu procházejících paketů se tedy provoz buďto povolí, nebo ne. Tímto jsme si zajistili adaptivnost našeho firewallu a z výše uvedeného testování lze tedy usoudit, že řešení projektu bylo pravděpodobně správně.

Literatura

1. *P4 Guide Installation*. [B.r.]. Dostupné také z: https://github.com/jafingerhut/p4-guide/blob/master/bin/README-install-troubleshooting.md.
2. *P4 Language Specification*. [B.r.]. Dostupné také z: https://p4.org/p4-spec/docs/P4-16-v1.2.2.html.
3. *P4 Language Tutorial*. 2017. Dostupné také z: https://opennetworking.org/wp-content/uploads/2020/12/P4_tutorial_01_basics.gslide.pdf.
4. *Cornell CS 6114: P4 Traffic Monitoring*. 2019. Dostupné také z: https://cornell-pl.github.io/cs6114/lecture07.html.
5. *BMv2 Simple Switch*. [B.r.]. Dostupné také z: https://github.com/p4lang/behavioral-model/blob/main/docs/simple_switch.md.
6. *P4-Learning*. [B.r.]. Dostupné také z: https://github.com/nsg-ethz/p4-learning.
7. *Runtime CLI*. [B.r.]. Dostupné také z: https://github.com/p4lang/behavioral-model/blob/main/docs/runtime_CLI.md.
8. *List of IP protocol numbers*. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné také z: https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers.
9. *ICMP Control Messages*. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné také z: https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol%5C#Control_messages.