ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Síťová aplikace v jazyce P4
Student:	Martin Čtrnáctý
Vedoucí:	Ing. Pavel Benáček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

1. Nastudujte přístupy k vyvažování výpočetní zátěže webových serverů.
2. Seznamte se s jazykem P4, kompilátorem pro softwarové běhové prostředí P4 a kompilátorem pro 100 Gb/s COMBO kartu, která je vybavena obvodem FPGA.
3. Implementujte aplikaci pro vyvažování výpočetní zátěže webových serverů v jazyce P4 podle provedeného návrhu.
4. Ověřte funkčnost a výkonnost aplikace pomocí běhového prostředí pro P4 programy. Diskutujte případné změny a rozšíření P4 kompilátoru pro 100 Gb/s COMBO kartu vybavenou obvodem FPGA .

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 26. září 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Síťová aplikace v jazyce P4

Martin Čtrnáctý

Katedra počítačových systémů
Vedoucí práce: Ing. Pavel Benáček, Ph.D.

7. ledna 2018

Poděkování

Rád bych touto formou poděkoval vedoucímu práce Ing. Pavlu Benáčkovi, Ph.D. za vstřícný přístup k mým dotazům a výborné vedení práce.

Dále děkuji také rodině a přátelům za podporu a trpělivost po celou dobu mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. ledna 2018 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Čtrnáctý, Martin. *Síťová aplikace v jazyce P4*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Bakalářská práce se zabývá možnostmi vyvažování zátěže za použití jazyka P4, který využívá principů softwarově definovaných sítí. Práce nejprve analyzuje dostupné algoritmy pro vyvažování zátěže, možnosti jazyka P4 a strukturu dat v druhé až čtvrté vrstvě síťového modelu ISO/OSI. Následně je navržený algoritmus implementován v jazyce P4 a jsou vytvořeny řídicí programy. Dále je testována funkčnost a výkonnost P4 aplikace v běhovém prostředí pro P4 programy. Závěrem je provedena diskuze o portaci P4 aplikace na kartu 100 Gb/s COMBO.

Klíčová slova vyvažování zátěže, jazyk P4, softwarově definované sítě

Abstract

The bachelor thesis addresses different ways of load balancing using P4 language, which is based on principles of Software-Defined Networks. The work provides an analysis of available load-balancing algorithms, P4 language options and data structure from the second to the fourth layer of the ISO/OSI network model. Subsequently, the proposed algorithm is implemented in P4 language together with control programs. It also provides tests of functionality

and performance in the P4 environment. Finally, there is a discussion about porting the application to a 100 Gb/s COMBO card.

Keywords load balancing, P4 language, Software-Defined Networking

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Vzhledem k rostoucím nárokům na rychlost a propustnost dnešních sítí neposkytuje klasický přístup k síťování dostatečně flexibilní infrastrukturu. V případě, že si při síťování nevystačíme se softwarovou implementací, vede příchod nového protokolu nebo lepšího přístupu k řešení určitého problému často k výměně samotného hardwaru. O změnu přístupu k síťování se pokouší koncept Software-Defined Networking, který se snaží zjednodušit správu sítě pomocí rozdělení síťové infrastruktury na řídicí logiku a datovou vrstvu. Na tomto konceptu postavená řešení dnes již obsluhují například dvacet procent provozu do internetu společnosti Google [1]. Jazyk P4, který řeší nevýhody Software-Defined Networking, potom přináší úplnou nezávislost na použitých protokolech a nabízí možnost si svoje síťové zařízení naprogramovat. Tento kód jakožto popis chování síťového prvku pak může být spuštěn na jakémkoli zařízení, které jazyk P4 podporuje.

Vyvažování zátěže je jednou z aplikací, které je v případě velkých datových toků vhodné implementovat jako hardware, ale zároveň je příhodné mít možnost ovlivnit chování tohoto hardwaru stejně jednoduše, jako je možné upravit chování softwaru. Práce má za cíl právě jednu takovou implementaci vyvažování zátěže zaměřenou na webový provoz připravit a otestovat. Práce spojuje možnosti abstrakce jazyka P4 s výhodami oddělené řídicí vrstvy konceptu Software-Defined Networking do jednoho funkčního zařízení.

První kapitola nejprve analyzuje známé přístupy a techniky k vyvažování zátěže, seznamuje s principy jazyka P4 včetně konceptu softwarově definovaných sítí a popisuje strukturu dat přenášených pomocí vybraných protokolů druhé až čtvrté vrstvy síťového modelu ISO/OSI. Potom popisuje principy protokolu HTTP a navrhuje vhodnou implementaci vzhledem k možnostem jazyka P4. Druhá kapitola seznamuje s primitivami jazyka P4 a použitými řešeními. Popisuje implementaci P4 aplikace a řídicích programů. Na závěr seznamuje s výsledky testování funkčnosti a výkonnosti celé P4 aplikace a diskutuje možnost portace P4 aplikace na kartu 100 Gb/s COMBO.

Kapitola 1

Analýza a návrh

V této kapitole jsou analyzovány dostupné způsoby vyvažování zátěže a principy jazyka P4 včetně problematiky softwarově definovaných sítí. Je zde vysvětlena datová struktura protokolů druhé až čtvrté vrstvy síťového modelu ISO/OSI použitých v P4 aplikaci vyvažování zátěže. Dále jsou zde popsány základní principy protokolu HTTP a je navržena vhodná aplikace pro vyvažování zátěže s využitím jazyka P4.

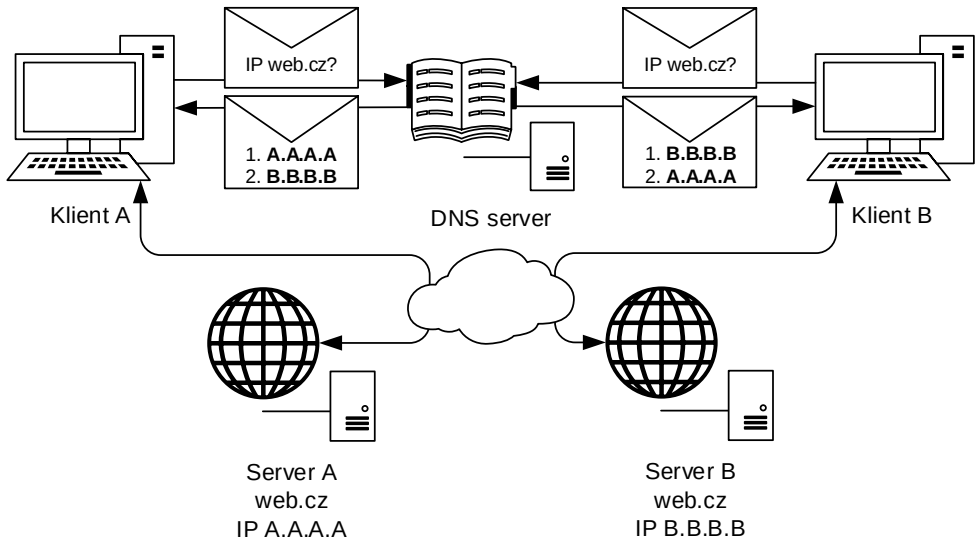
1.1 Vyvažování zátěže

Vyvažování zátěže dnes vzhledem ke stoupajícímu datovému provozu patří mezi důležité aplikace. Kromě rovnoměrného rozložení zátěže mezi více obsluhujících serverů může vyvažování zastávat i funkci překonání výpadku, kdy během nedostupnosti některého ze serverů přesměruje požadavky na ostatní servery a uživatelé tak nepocítí výpadek služby. Toho lze využít například i při potřebě údržby na některém ze serverů. Jednotlivé metody vyvažování lze na sebe vrstvit a vytvářet tak funkční celky, které umožňují rozkládat zátěž a překonávat výpadky.

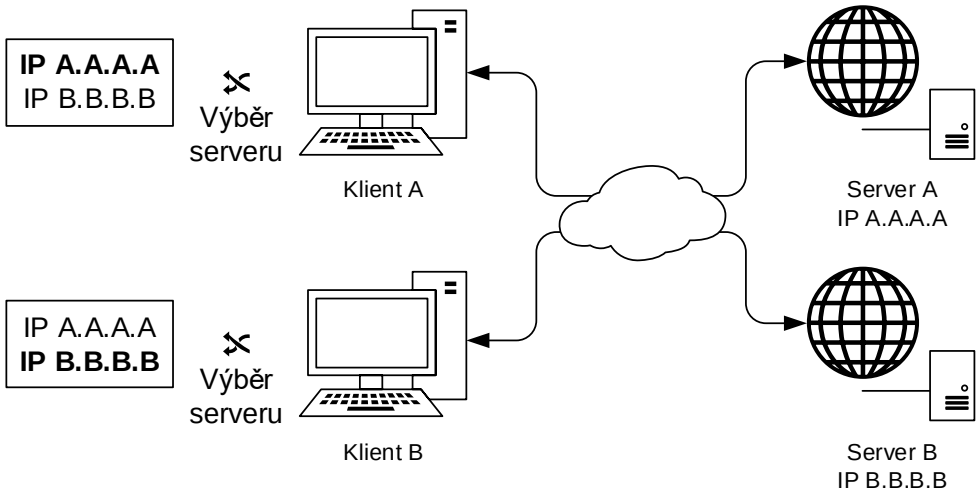
1.1.1 Round-robin DNS

Jedná se o metodu vyvažování, kdy se zátěž rozkládá již při překladu doménového jména na IP adresu. DNS server rotuje seznam možných IP adres tak, že se na prvním místě vždy střídají dostupné IP adresy. Klient si poté zpravidla vybere první jemu nabízenou IP adresu a dochází tak k rozložení zátěže, jak je možné vidět na obrázku 1.1. Pro moji práci je však nevhodná, neboť pracuje již na úrovni DNS serverů. [2]

Tato metoda má řadu nevýhod. Vzhledem k hierarchii DNS serverů, kde dochází k ukládání záznamů do mezipaměti, nemusí být vždy rozložení symetrické a lze ho jen těžko ovládat. Nedokáže také zohledňovat stav služby a její vytížení nebo celkovou kapacitu, protože rozděluje zátěž vždy rovným dílem



Obrázek 1.1: Vyvažování zátěže pomocí Round-robin DNS

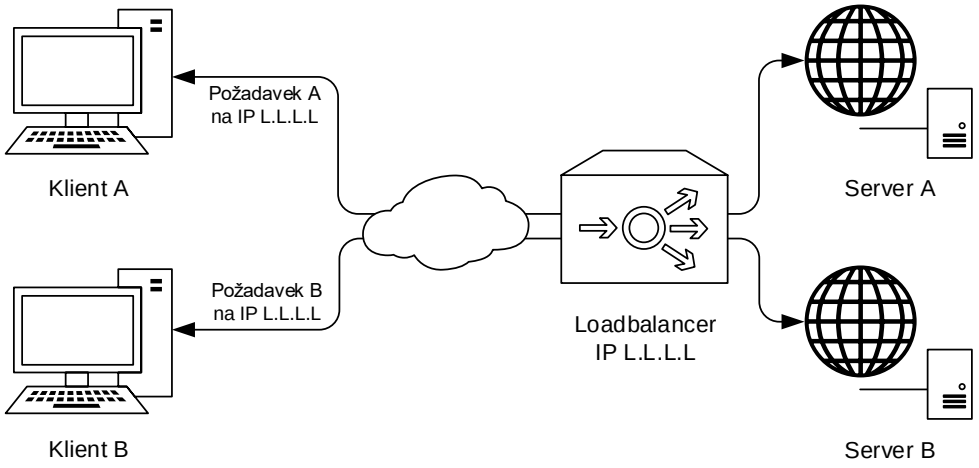


Obrázek 1.2: Vyvažování zátěže na straně klienta

mezi jednotlivé servery. I přes své nevýhody může najít uplatnění jako součást vícevrstevného rozkládání zátěže.

1.1.2 Vyvažování zátěže na straně klienta

Při této metodě je klientovi doručen seznam serverů, ze kterých si vždy vybere ten, se kterým bude komunikovat. Dle implementace je tak možné dávat někte-



Obrázek 1.3: Vyvažování zátěže na straně serveru

rým serverům větší prioritu či v případě nedostupnosti jednoho serveru zkusit jiný. Oproti round-robin algoritmu nedochází k ovlivnění mezipamětí DNS serverů. Princip vyvažování na straně klienta je možné vidět na obrázku 1.2. V běžném webovém provozu není tento algoritmus implementován a jeho využití by tedy vyžadovalo speciální aplikaci na straně klienta. Z uvedených důvodů není tento princip vyvažování vhodný.

1.1.3 Vyvažování zátěže na straně serveru

Jedná se o nejběžnější způsob vyvažování zátěže. Požadavky od klientů přijímá jedno zařízení, tzv. loadbalancer, který následně jednotlivé požadavky rozděluje mezi obsluhující servery. Server požadavek po vyřízení odešle zpět na loadbalancer, který pošle odpověď klientovi. Klient tedy komunikuje pouze s jediným zařízením. To umožňuje jednoduchou implementaci na straně klienta a může to představovat další vrstvu bezpečnosti, neboť klient nekomunikuje přímo se serverem. Princip vyvažování na straně serveru je možné vidět na obrázku 1.2.

Výhodou je rychlé a přesné řízení zátěže. Provoz můžeme regulovat pomocí široké škály parametrů, například zatížení, rychlosti odezvy nebo dostupnosti. Nevýhodou je velký datový tok přes loadbalancer a v případě selhání loadbalanceru nedostupnost celého systému.

Protože je nutné, aby veškerý provoz probíhal přes loadbalancer, softwarová implementace vyvažování zátěže nemusí v případě velkého zatížení stačit. Proto je vhodné, aby roli loadbalanceru zastával hardware. Pokud je však třeba změnit způsob rozkládání zátěže nebo přidat nový protokol, je vhodné mít možnost upravovat chování hardwarového loadbalanceru. K definici cho-

vání může být použit například jazyk P4, který umožňuje přesně definovat proces zpracování paketu.

1.2 Síťové protokoly

Protože P4 aplikace zasahuje do druhé až čtvrté vrstvy komunikace v síťovém modelu ISO/OSI [3], je nutné zjistit, jaká je struktura paketů v protokolech, které P4 aplikace z těchto vrstev používá. Rozložení jednotlivých vrstev modelu ISO/OSI, včetně příkladů protokolů/zařízení, ilustruje tabulka 1.1.

Tabulka 1.1: Model ISO/OSI

Č.	**Vrstva**	**Funkce**	**Příklad**
7	aplikační	komunikace s procesem	SMTP, HTTP
6	prezentační	prezentace dat a šifrování	JPEG, ASCII
5	relační	koordinace komunikace	NetBIOS, NFS
4	transportní	spojení	TCP, UDP
3	síťová	určení cesty a logická adresace	IPv4, IPv6
2	spojová	MAC a LLC - fyzická adresace	802.3 Ethernet
1	fyzická	přenosová média, signál	kabel, opakovač

1.2.1 Spojová vrstva

Spojová vrstva zajišťuje spojení mezi dvěma sousedními systémy. Stará se o nastavení parametrů linky a uspořádává data z fyzické vrstvy do jednotlivých celků nazývaných rámce. Oznamuje neopravitelné chyby a řídí přístup k fyzickému médiu. P4 aplikace podporuje z této vrstvy protokol IEEE 802.3 Ethernet [4], jehož uspořádání ilustruje tabulka 1.2.

Tabulka 1.2: Rámec protokolu IEEE 802.3 Ethernet

	Bity	
Bajty	0–15	16–31
0–3	cílová MAC adresa	
4–7	cílová MAC adresa	zdrojová MAC adresa
8–11	zdrojová MAC adresa	
12–15	802.1Q štítek	data
	data	
	kontrolní součet rámce	

Kontrolní součet tohoto protokolu je 32bitový a počítá se pomocí cyklického součtu. O jeho výpočet se v případě fyzické karty 100 Gb/s COMBO i běhového prostředí pro P4 programy stará přímo zařízení, a proto ho není nutné v P4 aplikaci řešit.

1.2.2 Síťová vrstva

Síťová vrstva zajišťuje směrování v síti a síťové adresování. Spojuje systémy, které spolu přímo nesousedí. Na této vrstvě pracují například protokoly IPv4 [5] a IPv6 [6]. P4 aplikace podporuje protokol IPv4, jehož uspořádání ilustruje tabulka 1.3.

Tabulka 1.3: Paket protokolu IPv4

	Bity				
Bajty	0–3	4–7	8–15	16–18	19–31
0–3	verze	délka hlavičky	typ služby	celková délka	
4–7	identifikace			fragmentace	offset
8–11	TTL		protokol	kontrolní součet	
12–15	zdrojová adresa				
16–19	cílová adresa				
	data				

1.2.3 Transportní vrstva

Transportní vrstva zajišťuje přenos dat mezi koncovými uzly. Jejím účelem je poskytovat takovou kvalitu přenosu, jakou požadují vyšší vrstvy. Z protokolů transportní vrstvy podporuje P4 aplikace protokoly TCP [7] (Transmission Control Protocol) a UDP [8] (User Datagram Protocol). Mezi těmito protokoly jsou výrazné rozdíly, které jsou popsány v následujících podsekcích. Protokol UDP je P4 aplikací podporován i přes její zaměření na webový provoz, protože P4 aplikace pracuje již na nízké úrovni sítě a je tedy možné využít P4 aplikaci i na jiné služby.

1.2.3.1 Protokol TCP

Protokol TCP zajišťuje přenos dat bez ztrát, bez duplikace a se zachováním pořadí přenášených paketů. Používá se například pro přenos souborů, webových stránek a emailů. Uspořádání paketu protokolu TCP ilustruje tabulka 1.4.

1.2.3.2 Protokol UDP

Protokol UDP zajišťuje přenos dat bez jakékoli záruky. Využívá se v aplikacích, u kterých by bylo na obtíž zpoždění způsobené čekáním na přenesení všech paketů a u kterých je možné ztráty řešit jiným způsobem. Používá se například pro DNS, VoIP a streamování videa. Hlavička je jednodušší než u protokolu TCP. Uspořádání celého paketu ilustruje tabulka 1.5.

Tabulka 1.4: Paket protokolu TCP

	Bity			
Bajty	0–3	4–6	7–15	16–31
0–3	zdrojový port			cílový port
4–7	číslo sekvence			
8–11	potvrzený bajt			
12–15	offset	rezervováno	příznaky	okénko
16–19	kontrolní součet			urgentní ukazatel
	data			

Tabulka 1.5: Paket protokolu UDP

	Bity	
Bajty	0–15	16–31
0–3	zdrojový port	cílový port
4–7	délka	kontrolní součet
	data	

1.2.4 Kontrolní součty

Pro protokoly IPv4, UDP a TCP musí P4 aplikace umět počítat kontrolní součty, které slouží k ověření, zda je informace úplná a zda při jejím přenosu nedošlo k chybě. Při přijímání paketu se kontrolní součet vypočítá a porovná s tím, který byl přijat v paketu. Pokud souhlasí, paket s největší pravděpodobností nebyl během cesty poškozen. Při odesílání paketu se kontrolní součet vypočítá a uloží do dané části hlavičky paketu. Pro výpočet kontrolního součtu je třeba vědět, jaké položky mají být součástí kontrolního součtu a jakým způsobem se kontrolní součet počítá.

Součástí kontrolního součtu u protokolu IPv4 je pouze samotná hlavička tohoto protokolu. Součástí kontrolního součtu u protokolů TCP a UDP je pseudo hlavička, hlavička a data. Pseudo hlavička se vytváří pouze pro výpočet kontrolního součtu a nikdy se nikam neodesílá. Pseudo hlavičku ilustruje tabulka 1.6.

Tabulka 1.6: Pseudo hlavička

	Bity			
Bajty	0–7	8–15	16–18	19–31
0–3	zdrojová IPv4 adresa			
4–7	cílová IPv4 adresa			
8–11	zarovnání	protokol z IPv4	délka paketu	

Samotný výpočet kontrolního součtu poté probíhá následovně:

1. Všechny položky, které mají být součástí kontrolního součtu, seřadíme za sebe a rozdělíme po šestnácti bitech.
2. Uděláme součet všech položek.
3. Přičteme případný přenos nad šestnáct bitů.
4. Provedeme bitovou negaci součtu.
5. V případě, že je výsledek nula, přičteme jedničku.

1.2.5 Protokol HTTP

Webový provoz zpracovávaný v P4 aplikaci využívá protokol HTTP/1.0 nebo HTTP/1.1 nad protokolem TCP. Pro vyvažování zátěže je důležité zejména to, jakým způsobem vytváří spojení a jak odesílá požadavky. Ve verzi HTTP/1.0 je pro každý požadavek vytvořeno nové TCP spojení se serverem a po obsloužení jednoho požadavku dojde k ukončení spojení. Jelikož to znamená velkou režii na vytváření a ukončování spojení, verze HTTP/1.1 již umožňuje obsloužit více požadavků v rámci jednoho spojení. Tato spojení se ale obvykle udržují jen na krátkou dobu v řádu několika vteřin. Poté dojde k ukončení spojení a při dalším požadavku se vytváří nové.

1.3 Software-Defined Networking

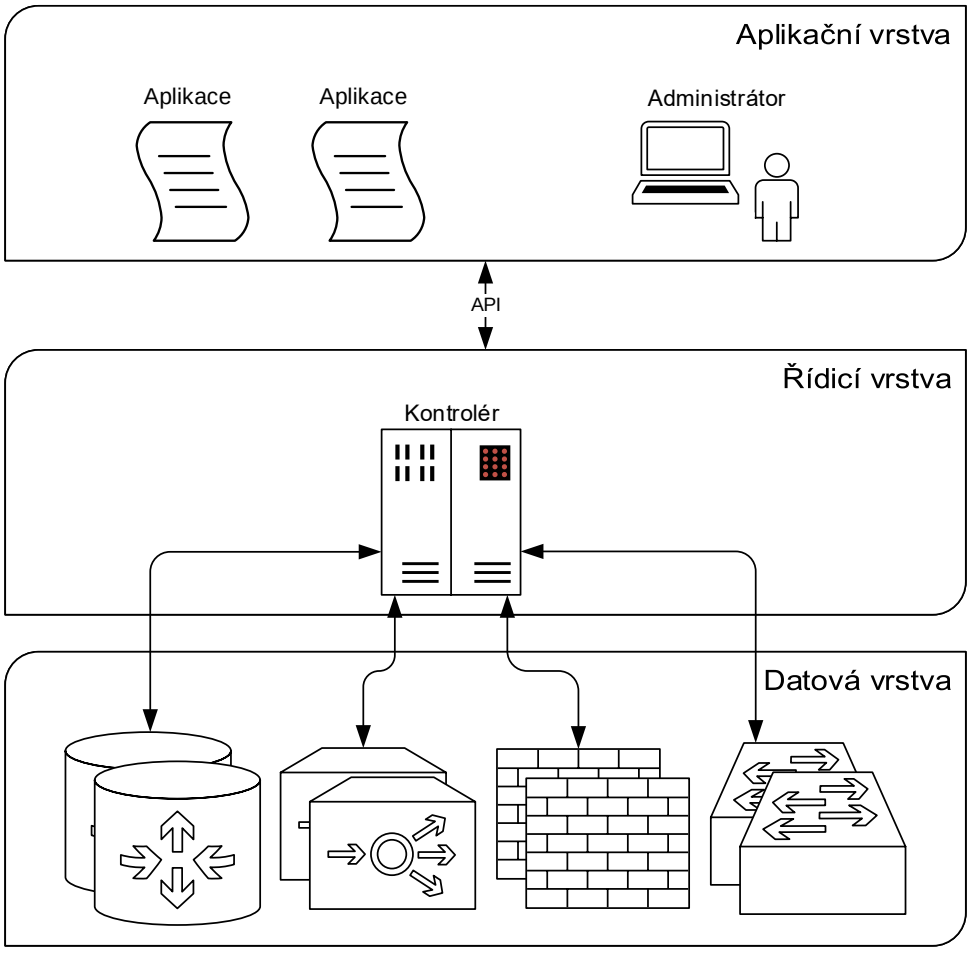
Z důvodu rostoucí rychlosti a složitosti dnešních sítí je třeba jednoduše a rychle spravovat rozsáhlou síťovou architekturu, což může být v síti s hardwarem od různých výrobců problém. Každé zařízení je nutné konfigurovat zvlášť a k úpravě chování v celé síti je třeba překonfigurovat velké množství zařízení. Koncept Software-Defined Networking [9] se tedy snaží rozdělit řídicí logiku a datovou vrstvu síťové infrastruktury. Řídicí logika má potom na starosti více síťových prvků, kterým určuje, jak se mají chovat. Hardwarové prvky neobsahují žádný řídicí software a v případě, že narazí na provoz, se kterým si neví rady, odešlou ho k analýze do řídicí vrstvy. Rozdělení na jednotlivé vrstvy je možné vidět na obrázku 1.4 a přináší tyto výhody:

Přímá konfigurovatelnost
: Možné díky oddělení řídicí a datové vrstvy.

Abstrakce
: Pro většinu činností nemusíme znát přesnou topologii sítě.

Centralizace
: Přináší výhody při správě sítě, řízení toku dat v síti atd.

Otevřený standard
: Přináší nezávislost na výrobci síťového hardwaru.

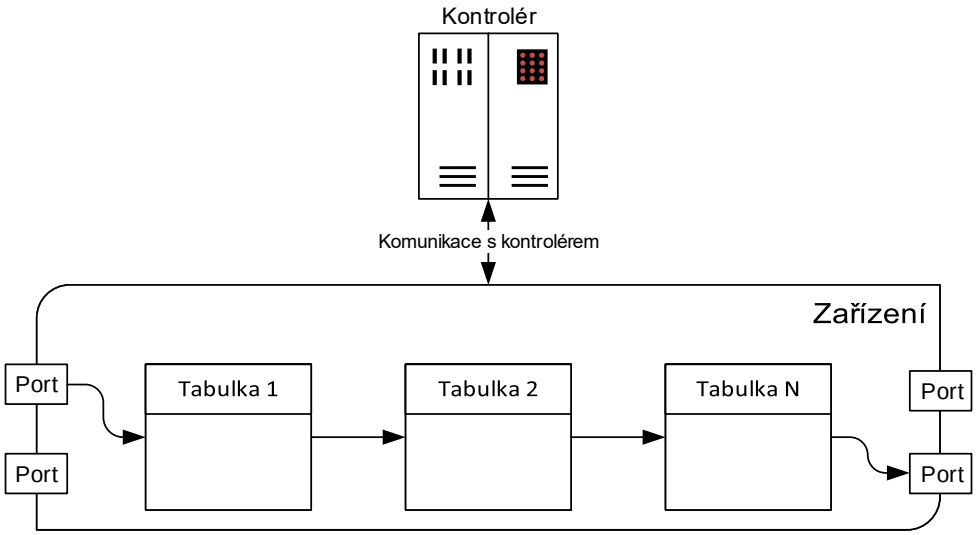


Obrázek 1.4: Architektura SDN, převzato z [9]

API
: S kontrolérem může uživatel nebo aplikace komunikovat přes standardizované API z aplikační vrstvy.

1.3.1 OpenFlow

Protokol OpenFlow sloužící pro komunikaci mezi řídicí a datovou vrstvou je jednou z možností, jak realizovat koncept SDN. Základním bodem zpracování provozu je tabulka, která se plní pravidly, které nastavuje OpenFlow kontrolér. Každé pravidlo pak definuje akci, která se má na daný paket aplikovat. Tímto způsobem může kontrolér realizovat různé algoritmy pro zpracování paketu. Schéma zpracování provozu pomocí tabulek ukazuje obrázek 1.5. Nevýhodou OpenFlow je omezený počet podporovaných protokolů a akcí, které jsou pevně definovány v OpenFlow standardu. Přidání nového protokolu znamená větši-



Obrázek 1.5: Zpracování provozu pomocí tabulek, převzato z [9]

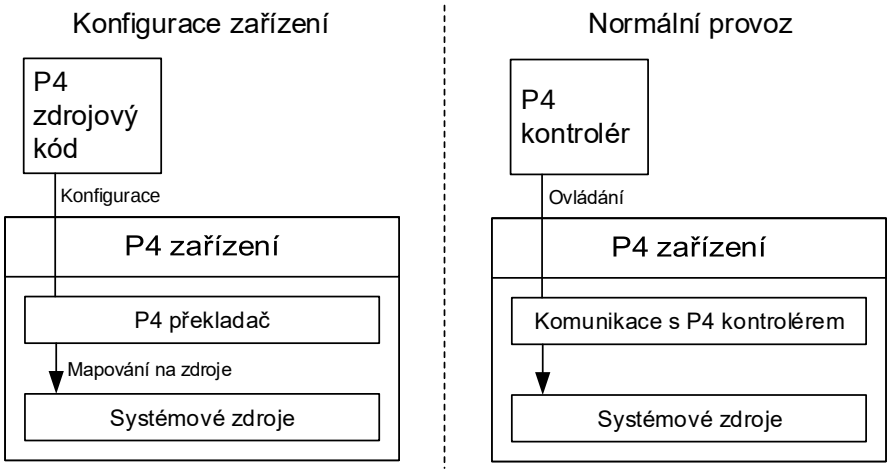
nou výměnu zařízení. Tato omezení se snaží odstranit jazyk P4.

1.4 Jazyk P4

Jazyk P4 [10] neboli **P**rogramming **P**rotocol-independent **P**acket **P**rocessors umožňuje popisovat chování síťového prvku nezávisle na cílové architektuře a doplňuje koncept Software-Defined Networking. Jazyk P4 existuje aktuálně ve specifikacích $P4_{14}$ a $P4_{16}$. Používána v implementaci a popisována v návrhu je verze $P4_{14}$ [11], která byla v době implementace považována za stabilní, zatímco $P4_{16}$ byla ještě ve fázi návrhu.

V jazyce P4 je možné popsat, jakým způsobem má být zpracováván síťový provoz, aniž by bylo nutné zajímat se o to, jestli bude P4 aplikace softwarově simulována nebo bude například vystavěna v programovatelném hradlovém poli. Kompilace programu probíhá v momentě jeho nahrávání do daného zařízení a může tedy běžet na libovolném zařízení, které podporuje P4 (např. CPU, GPU, síťový procesor, programovatelné hradlové pole, ...). To umožňuje vytváření síťových prvků, u kterých rozhoduje o podporovaných protokolech pouze softwarová výbava a které mohou být v případě využití programovatelných hradlových polí i velice rychlé. [12]

P4 zařízení nejprve pracuje v módu konfigurace, během kterého se do něj nahrává program. Poté se dostává do normální fáze, kdy zpracovává síťový provoz, jak ukazuje obrázek 1.6. Stejně jako u protokolu OpenFlow pak zpracování probíhá pomocí tabulek, jak je možné vidět na obrázku 1.5. Jakým způsobem se bude zařízení chovat, to je určováno kontrolérem, který zasa-



Obrázek 1.6: Režimy práce P4 zařízení, převzato z [12]

huje do tabulek zařízení, čímž rozhoduje o tom, jaké akce budou nad určitým provozem prováděny.

Samotný $P4_{14}$ program se pak dělí na části:

- specifikace hlaviček protokolů,
- parsování dat z paketu,
- tabulky,
- akce,
- kontrolní část.

Tyto části specifikují, jakým způsobem bude provoz zpracováván, a jsou vysvětleny na příkladech v následujících podsekcích.

1.4.1 Specifikace hlaviček protokolů

Tato část určuje, jaké hlavičky mají protokoly, které jsou extrahovány z paketu. Je možné také definovat vlastní struktury, které se hodí, pokud je potřeba ukládat nějaké další informace potřebné ke zpracování paketu.

Ukázka definice hlavičky protokolu IEEE 802.3 Ethernet:

```
header_type ethernet_t {
  fields {
    dstAddr : 48;
    srcAddr : 48;
    etherType : 16;
  }
}
```

1.4.2 Extrahování dat z paketu

Proces popisuje, v jakém pořadí se mají extrahovat data z příchozích paketů a jak reagovat na určité hodnoty v hlavičkách. Také se zde specifikují akce spouštěné v případě výjimky při extrahování. Zápis je ve formátu konečného automatu, kde se specifikuje nejen extrakce, ale také přechodová podmínka, zda se bude pokračovat v parsování dalšího protokolu, nebo zda začne zpracování.

Ukázka definice parseru protokolu IPv4:

```
parser parse_ipv4 {
  extract(ipv4);
  return select(latest.fragOffset, latest.protocol) {
    6 : parse_tcp;
    17 : parse_udp;
    default: ingress;
  }
}
```

1.4.3 Tabulky

Tabulky hrají v jazyce $P4_{14}$ významnou roli. Využívají se totiž pro zpracování paketu podle zadaných pravidel. Každá tabulka má definováno jedno nebo více polí, kterými se v ní vyhledává. Hodnoty polí právě zpracovávaného paketu se srovnávají s hodnotami odpovídajících polí u pravidel v tabulce. U každého pole je specifikován takzvaný algoritmus vyhledávání, který určuje, jak budou hodnoty srovnávány. Algoritmy vyhledávání mohou být následující:

Shoda (exact)
: Hodnota pole vyextrahovaného z paketu se shoduje s hodnotou pole v tabulce.

Shoda po vymaskování (ternary)
: Hodnoty polí se shodují po aplikování masky. Maskuje se pomocí logického AND a každé pravidlo v tabulce má svoji vlastní masku.

Nejdelší shoda předpony (lpm)
: Najde se pravidlo, ve kterém se pole vyextrahované z paketu nejdéle zleva shoduje s polem uvedeným v tabulce.

Rozsah (range)
: Hodnota pole z paketu je z rozsahu nastaveného u pravidla v tabulce.

Validita (valid)
: Kontroluje, jestli je pole validní, tedy zda je naplněno daty. Lze použít pouze na pole extrahované z paketu.

Může dojít i k tomu, že se najde více pravidel, která splňují zadaná kritéria. Poté přichází na řadu priority nastavené u jednotlivých pravidel.

Pokud je v tabulce nalezena shoda s pravidlem, spustí se akce, která je u něj nastavená. Každé pravidlo v tabulce má uložené parametry, se kterými je akce spuštěna. Pokud není v tabulce nalezeno odpovídající pravidlo, provede se výchozí akce, která je u tabulky nastavená.

Ukázka definice tabulky:

```
table port {
  reads {
    standard_metadata.ingress_port : exact;
    ipv4.ttl : range;
    loadbalance_metadata.l4packet : exact;
  }
  actions {to_server; to_client; _drop;}
  size : 512;
}
```

1.4.4 Akce

Akce jsou určité skupiny prováděných operací, které obvykle popisují složitější funkcionalitu. Mohou obsahovat buď jiné akce, nebo takzvané primitivní akce. Primitivní akce jsou dále nedělitelné a jejich funkcionalita je definována ve standardu. Samotné akce nemohou obsahovat žádné podmínky, čímž neumožňují vytvářet jakoukoli řídicí logiku. Složitější funkcionalita je v $P4_{14}$ realizována pomocí tabulek a řídicího programu.

Ukázka definice akce:

```
action update_ttl() {
  modify_field(ipv4.ttl, ipv4.ttl - 1);
}
```

1.4.5 Kontrolní část

Kontrolní část rozhoduje o tom, kdy a jaká tabulka se aplikuje na právě zpracovávaný paket, čímž zastává hlavní logiku P4 aplikace. Tabulky lze aplikovat těmito způsoby:

Sekvenčně
: Jedna po druhé bez vyhodnocování podmínek.

Předchozí akce
: Podle akce vyvolané předchozí tabulkou.

„Hit“ nebo „Miss“
: Podle výsledku hledání v předchozí tabulce.

Podmínka
Podle porovnání konkrétního výrazu.

Ukázka kontrolní části:

```
control ingress{
  apply(port){
    to_server{
      apply(connection){
        miss{
        apply(hash);
        }
      }
      apply(server);
    }
  }
}
```

1.5 Behavioral model

Behavioral model [13] je prostředí obsahující nástroje pro simulaci běhu P4 zařízení, komunikaci s P4 aplikací a ladění P4 aplikace. Toto prostředí je P4 komunitou stále vyvíjeno a obsahuje tak chyby, které práci komplikovaly. Následující podsekce popisují součásti tohoto modelu, které jsou použity v P4 aplikaci.

1.5.1 Nanomsg klient

Jedná se o program implementovaný v jazyce Python, který umožňuje přijímat zprávy od P4 aplikace přes nanomsg protokol. Mohou to být jak zprávy, které slouží k ladění P4 aplikace, tak i zprávy, které P4 aplikace odesílá za účelem upozornění na situaci, na kterou bude potřeba reagovat, například přidáním pravidla do tabulky. V implementaci je nanomsg klient použit jako modul řídicího programu v Pythonu a jsou volány jeho konkrétní funkce, které jsou zapotřebí pro přijímání a potvrzování zpráv. Také byl použit k ladění P4 aplikace během implementace. S P4 aplikací komunikuje pomocí IPC rozhraní, které je potřeba specifikovat při spuštění simulace.

1.5.2 Runtime CLI

Tento program, který je taktéž implementovaný v jazyce Python, umožňuje kompletní ovládání běhového prostředí P4 aplikace. Zaměřuje se zejména na kontrolu a nastavování tabulek. Má vlastní rozhraní příkazového řádku, které podporuje doplňování příkazů. V implementaci je použit jako modul v obou

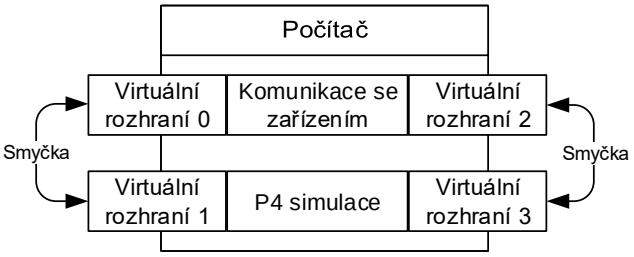
kontrolních programech na přidávání a změnu pravidel v tabulkách P4 aplikace.

1.5.3 Zařízení simple-switch

Projekt behavioral model obsahuje několik zařízení, která představují spustitelné prostředí simulace P4 programu. Jednotlivá zařízení se mezi sebou liší podporovanými funkcemi. Pro implementaci bylo vybráno právě zařízení simple-switch, protože obsahuje nejbohatší implementaci primitiv jazyka P4 a bude tedy k budoucímu vývoji nejvhodnějším nástrojem.

1.5.4 Virtuální rozhraní

Protože P4 aplikace komunikuje po síti, je třeba vytvořit rozhraní, ke kterým bude připojena. V případě simulace se jedná o virtuální rozhraní, která jsou vždy ve dvojici vzájemně propojena smyčkou tak, že data, která odešleme do prvního rozhraní, vyjdou z druhého rozhraní ve dvojici, jak je znázorněno na obrázku 1.7. To je nutné, neboť v případě, že je potřeba komunikovat s P4 aplikací z počítače, je třeba, aby data odeslaná z nějakého rozhraní ven dorazila do P4 aplikace, která běží uvnitř počítače, a naopak. Vytvoření rozhraní obstarává jeden ze skriptů obsažených v behavioral modelu.



Obrázek 1.7: Způsob zapojení virtuálních rozhraní

1.6 P4 překladač

Implementace je překládána pomocí P4 překladače [14] vytvářeného komunitou jazyka P4. Tento překladač podporuje verze jazyka $P4_{14}$ a $P4_{16}$. Umí překládat do několika formátů:

p4c-bm2-ss
: Generuje popis P4 programu v jazyce JSON, který behavioral model umí interpretovat.

p4c-ebpf
: Generuje kód v jazyce C pro obsluhu požadavků přímo z linuxového jádra.

p4test
: Formát pro testovací účely a ladění překladače.

p4c-graphs
: Formát pro generování grafové reprezentace P4 aplikace.

Pro překlad implementace je používán formát pro behavioral model a verzi jazyka $P4_{14}$.

1.7 Návrh

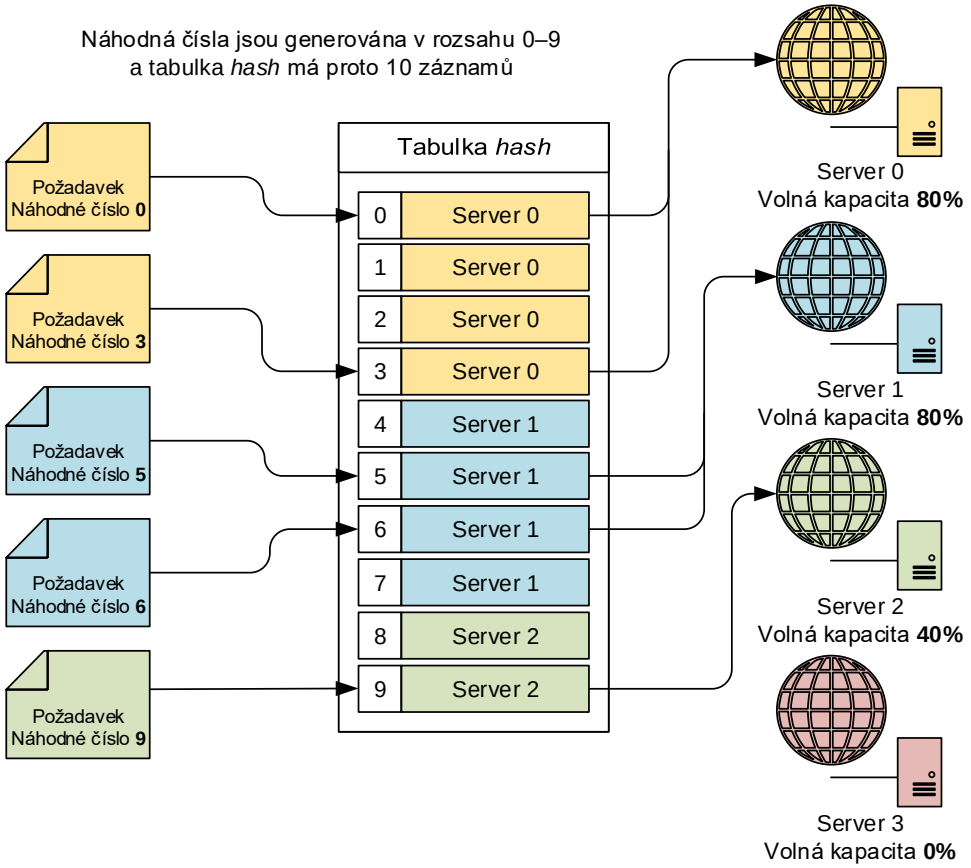
P4 aplikace na vyvažování zátěže je zařízení, které má dvě rozhraní. Jedno je připojeno do sítě klientů a druhé do sítě, kde jsou umístěny jednotlivé servery. Z klientské sítě přichází na klientské rozhraní požadavky, které je třeba rozdělovat mezi servery. Od serveru poté přichází odpovědi, které je potřeba doručit klientům. P4 aplikace je primárně určena pro vyvažování webového provozu, ale z podstaty jejího fungování je možné využít ji i v jiném prostředí.

1.7.1 Rozdělení požadavků

S dostupnými prostředky jazyka $P4_{14}$ je potřeba rozdělit provoz na co nejvíce možných částí, kdy každá může být přidělena ke zpracování jinému serveru. P4 aplikace používá bezstavové zpracování a musí si tak pro rozdělení vystačit s informacemi, které nese aktuálně zpracovávaný paket. Jazyk $P4_{14}$ umožňuje i stavové zpracování, kdy je možné ukládat si do registrů určité informace z paketů, které předcházely, avšak taková implementace již není triviální a může ovlivňovat výkonnost celého řešení. Jako vhodné rozdělení byl tedy zvolen prvek komunikace, který má unikátní trojici údajů:

- zdrojová IPv4 adresa,
- zdrojový TCP/UDP port,
- použitý transportní protokol (TCP nebo UDP).

Tento prvek nezaručuje rozdělení provozu na jednotlivé HTTP požadavky. K tomu by bylo zapotřebí analyzovat vyšší než čtvrtou vrstvu modelu ISO/OSI. To však práce neprovádí, neboť jazyk P4 se neumí vypořádat s případnou fragmentací TCP provozu a musel by tak být schopen pracovat s několika bajty aplikačního protokolu. Výše zmíněná trojice údajů však dokáže rozdělit provoz na malé skupiny požadavků, protože zdrojová IPv4 adresa, společně se zdrojovým portem, identifikují jednotlivého uživatele a spojení je i v případě



Obrázek 1.8: Vyvažování zátěže pomocí pseudonáhodných čísel

HTTP/1.1 většinou udržováno pouze v řádu vteřin. Maximální délku spojení je možné ovlivnit díky tomu, že servery budou nasazovány ve známém prostředí a administrátorem, který může případně upravit jejich konfiguraci. Jakmile je spojení ukončeno, při navazování nového již bude pravděpodobně použit jiný zdrojový port. Tím je provoz rozdělen na několikavteřinová spojení od jednotlivých uživatelů. Problém by mohly představovat proxy servery, neboť dokáží sdružovat požadavky od více klientů za jednu IPv4 adresu.

1.7.2 Vyvažování zátěže

Z důvodu využití jazyka $P4_{14}$ byla zvolena metoda vyvažování zátěže na straně serveru a rozkládání zátěže je prováděno na základě informací o volných prostředcích, které jsou získávány od serverů. Z příchozího paketu se vygeneruje pseudonáhodné číslo z určitého rozsahu, pomocí kterého je vybrán server, který má obsloužit daný provoz. Server je vybrán z tabulky, která má stejný rozsah, v jakém se generují pseudonáhodná čísla, a jsou v ní zastoupeny jednot-

livé servery dle poměru volných kapacit. Rozdělení požadavků pomocí tabulky a pseudonáhodných čísel znázorňuje obrázek 1.8.

Pokud jsou například dva servery, které hlásí stejnou volnou kapacitu, obsahuje tabulka pro oba dva servery stejné množství pravidel. V případě, že první server hlásí dvakrát větší volnou kapacitu než server druhý, obsahuje tabulka dvakrát více pravidel prvního serveru než serveru druhého. Tím je dvakrát větší šance, že na základě pseudonáhodně vygenerovaného čísla bude vybrán server první místo serveru druhého. Zároveň je možné nastavit volnou kapacitu na nulu, což zajistí, že nebudou na server posílány žádné požadavky.

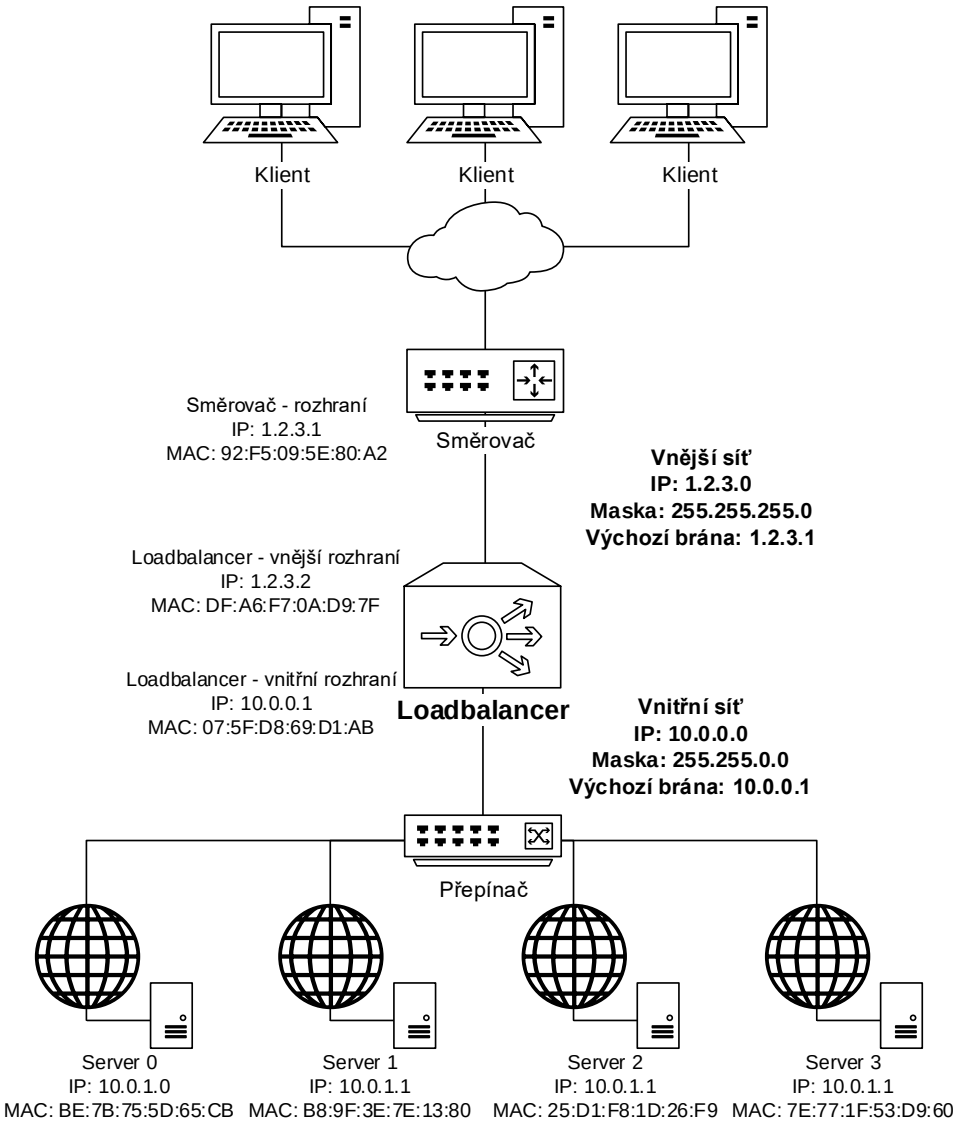
1.7.3 Síťové řešení

Protože P4 aplikace komunikuje po síti, bylo potřeba navrhnout, jakým způsobem bude v síti pracovat. P4 aplikace je do sítě připojena dvěma rozhraními. Jedním do vnější sítě, druhým do sítě vnitřní.

Na vnějším rozhraní má adresu sítě, ze které přijímá požadavky. Většinou tedy veřejnou IPv4 adresu. Na toto rozhraní jsou klienti schopni doručit svoje pakety s požadavky. Ve vnější síti se také nachází směrovač, který je schopen směrovat pakety z vnějšího rozhraní loadbalanceru ke klientům.

Vnitřní rozhraní má adresu z rozsahu interní sítě, kde jsou umístěny servery. Každý server má jedno rozhraní připojené do vnitřní sítě s IPv4 adresou z rozsahu vnitřní sítě. Každé rozhraní má ještě svoji MAC adresu. Mezi servery a vnitřním rozhraním se musí nacházet přepínač, který zajistí možnost připojení více serverů. Příklad takového uspořádání ilustruje obrázek 1.9. Teoreticky by nemusely být servery připojeny přímo do sítě, kde je připojeno vnitřní rozhraní loadbalanceru. Mohly by být v jiné síti, do které by bylo možné pakety směrovat pomocí směrovačů. Konfigurace P4 aplikace by to umožnila, ale pro jednoduchost tuto možnost nepopisuji.

V případě příchodu paketu od klienta na vnější rozhraní určí P4 aplikace server, na který má být požadavek přeposlán, a podle toho upraví cílovou IPv4 adresu a cílovou MAC adresu. Podle adresy svého vnitřního rozhraní upraví také zdrojovou MAC adresu a odešle paket na vnitřní rozhraní. V případě příchodu paketu na vnitřní rozhraní P4 aplikace implicitně předpokládá, že se jedná o paket pro klienta. Změní proto zdrojovou IPv4 adresu a zdrojovou MAC adresu na odpovídající adresy svého vnějšího rozhraní. Upraví také cílovou MAC adresu na adresu routeru ve vnější síti a odešle paket na vnější rozhraní. Aby servery věděly, kam mají poslat pakety s odpovědí, musí mít ve své směrovací tabulce nastavené pravidlo, které zajistí, že všechny sítě, které mohou poslat pakety na vnější rozhraní loadbalanceru, budou směrovány na vnitřní rozhraní loadbalanceru. Toho lze nejlépe dosáhnout označením vnitřního rozhraní loadbalanceru jako výchozí brány.



Obrázek 1.9: Příklad síťového řešení

Kapitola 2

Implementace

V této kapitole je popsán způsob implementace P4 aplikace a problémy, které při implementaci vznikly. Implementovat bylo třeba samotný loadbalancer v jazyce $P4_{14}$ a také dva kontrolní programy. Jeden z kontrolních programů ovládá rozdělování zátěže, druhý aktualizuje tabulku známých spojení.

2.1 $P4_{14}$ aplikace

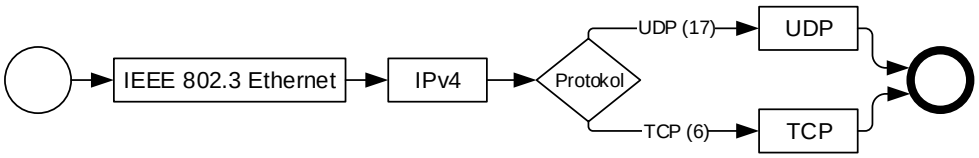
V této části se práce zabývá samotnou implementací P4 aplikace v jazyce $P4_{14}$. Vysvětluje způsob implementace jednotlivých částí a jejich zapojení do P4 aplikace jako celku.

2.1.1 Hlavičky a pomocné struktury

Jako první je třeba v P4 aplikaci definovat jednotlivé hlavičky protokolů IEEE 802.3 Ethernet [4], IPv4 [5], TCP [7] a UDP [8] dle jejich specifikace. Také je potřeba definovat pomocnou strukturu, do které jsou ukládána data, která jsou zapotřebí ke zpracování paketu.

2.1.2 Parsování

Po definici hlaviček protokolů je pomocí parseru rozložen příchozí paket. Nejprve se z paketu získávají hlavičky rámce IEEE 802.3 Ethernet a poté hlavičky



Obrázek 2.1: Parsování v P4 aplikaci

protokolu IPv4. Pak se parser na základě informací z hlavičky IPv4 protokolu rozhodne, jestli bude pokračovat protokolem UDP nebo TCP, a provede poslední parsování. Průběh parsování ilustruje obrázek 2.1. Během parsování jsou ukládány informace o zdrojovém a cílovém portu do pomocné struktury, protože k nim je třeba přistupovat nezávisle na protokolu.

2.1.3 Výpočet kontrolních součtů

Kontrolní součty se v jazyce $P4_{14}$ sestavují pomocí tří kroků. Nejprve se definuje struktura zvaná `field_list`, která obsahuje jednotlivé položky, s nimiž bude potřeba pracovat. Tento `field_list` se chová jako jedna dlouhá sekvence bitů, jež obsahuje hodnoty položek, které jsou do něj zařazeny poskládané za sebou.

Příklad definice `field_list`:

```
field_list ipv4_checksum_list {
  ipv4.version;
  ipv4.ihl;
  ipv4.diffserv;
  ipv4.totalLen;
  ipv4.identification;
  ipv4.flags;
  ipv4.fragOffset;
  ipv4.ttl;
  ipv4.protocol;
  ipv4.srcAddr;
  ipv4.dstAddr;
}
```

Nad tímto listem se poté definuje `field_list_calculation`, který slouží pro nastavení parametrů výpočtu kontrolního součtu (typ algoritmu, vstupní `field_list` struktury a výstupní šířka). Implementací používaný typ simulace v prostředí behavioral model aktuálně podporuje pouze metodu `csum16`, kterou lze použít pro kontrolní součty používaných protokolů.

Příklad definice `field_list_calculation`:

```
field_list_calculation ipv4_checksum {
  input {
    ipv4_checksum_list;
  }
  algorithm : csum16;
  output_width : 16;
}
```

Posledním krokem je specifikace `calculated_field`, která označuje, že některá z položek hlavičky je počítaná. Tato hlavička se při parsování paketu

ověří a při deparsování paketu znovu spočítá. Aktuální překladač nepodporuje výjimky při parsování paketu, takže sice nevalidní kontrolní součet detekuje, ale neumí vyvolat výjimku při parsování. Tento nedostatek by se dal vyřešit například spočítáním kontrolního součtu v programu zvlášť a jeho ověřením pomocí podmínky.

Příklad definice `calculated_field`:

```
calculated_field ipv4.headerChecksum  {
  verify ipv4_checksum;
  update ipv4_checksum;
}
```

Zajímavostí je, že u protokolu UDP je informace o délce paketu zahrnuta v kontrolním součtu dvakrát. Jednou je to v pseudo hlavičce, podruhé v hlavičce samotného UDP protokolu. Protokol TCP informaci o délce paketu v hlavičce nemá, takže u něj se tato anomálie nevyskytuje. Během implementace byla také v překladači chyba, která způsobovala, že se ve `field_list` neinterpretovala položka `payload` zastupující data. P4 aplikace tedy odesílala pakety se správným kontrolním součtem pouze v případě, že byly bez datové části, nebo datovou část tvořily samé nuly. To, společně s výše uvedenou anomálií, způsobilo během implementace nejeden problém. Naštěstí byla implementace překladače později komunitou opravena a nyní fungují kontrolní součty v pořádku.

2.1.4 Výpočet pseudonáhodného čísla

V případě příchodu paketu, pro který nebylo nalezeno pravidlo v tabulce *spojení*, je potřeba jej přiřadit některému ze serverů. To se děje pomocí tabulky *hash*, která má fixní počet pravidel a ve které jsou dle poměru aktuálně volné kapacity zastoupeny jednotlivé servery. V této tabulce se vyhledává pomocí pseudonáhodně vygenerovaného čísla, které je získáno podobně jako kontrolní součet. Nejprve je tedy definován `field_list` obsahující zdrojovou IPv4 adresu, zdrojový port a použitý protokol. Nad tímto listem je definován `field_list_calculation` metodou `csum16`. V P4 aplikaci je pak volána funkce `modify_field_with_hash_based_offset`, která provede výpočet pomocí vzorce `(base + (hash_value % size))`, kde `base` je minimální hodnota z intervalu, ve kterém má být číslo vygenerováno, a `size` je počet prvků v tomto intervalu. Tím je získána hodnota z rozsahu `base` a `(base + size - 1)`. Hodnota `base` je v případě implementované P4 aplikace 0 a hodnota `size` je rovna počtu záznamů v tabulce *hash*.

Jedná se tedy opravdu o pseudonáhodné číslo, které je však dostačující k tomu, aby se mezi jednotlivými aktualizacemi tabulky *hash* zátěž rozkládala mezi servery s volnou kapacitou. Hlavní rozkládání zátěže provádí program, který do tabulky *hash* vhodně dosazuje servery s volnou kapacitou.

2.1.5 Odesílání zpráv

Po přiřazení neznámého paketu některému ze serverů je třeba informovat řídicí program, který zajistí přidání pravidla do tabulky *spojení*. To se děje pomocí primitivní funkce `generate_digest`, jíž je předáno jméno listu, který je třeba odeslat řídicímu programu. V P4 aplikaci je proto vytvořen list obsahující zdrojovou IPv4 adresu, zdrojový port a protokol. Ten je následně odeslán do řídicího programu.

2.1.6 Tabulky a akce

Samotná kontrolní část P4 aplikace, tedy část rozhodující o tom, kdy a jaká tabulka se aplikuje na právě zpracovávaný paket, je znázorněna na obrázku 2.2, kde jsou modře vyznačeny tabulky. V následujícím textu jsou popsány jednotlivé tabulky, které kontrolní část spouští, a akce, které následují.

2.1.6.1 Port

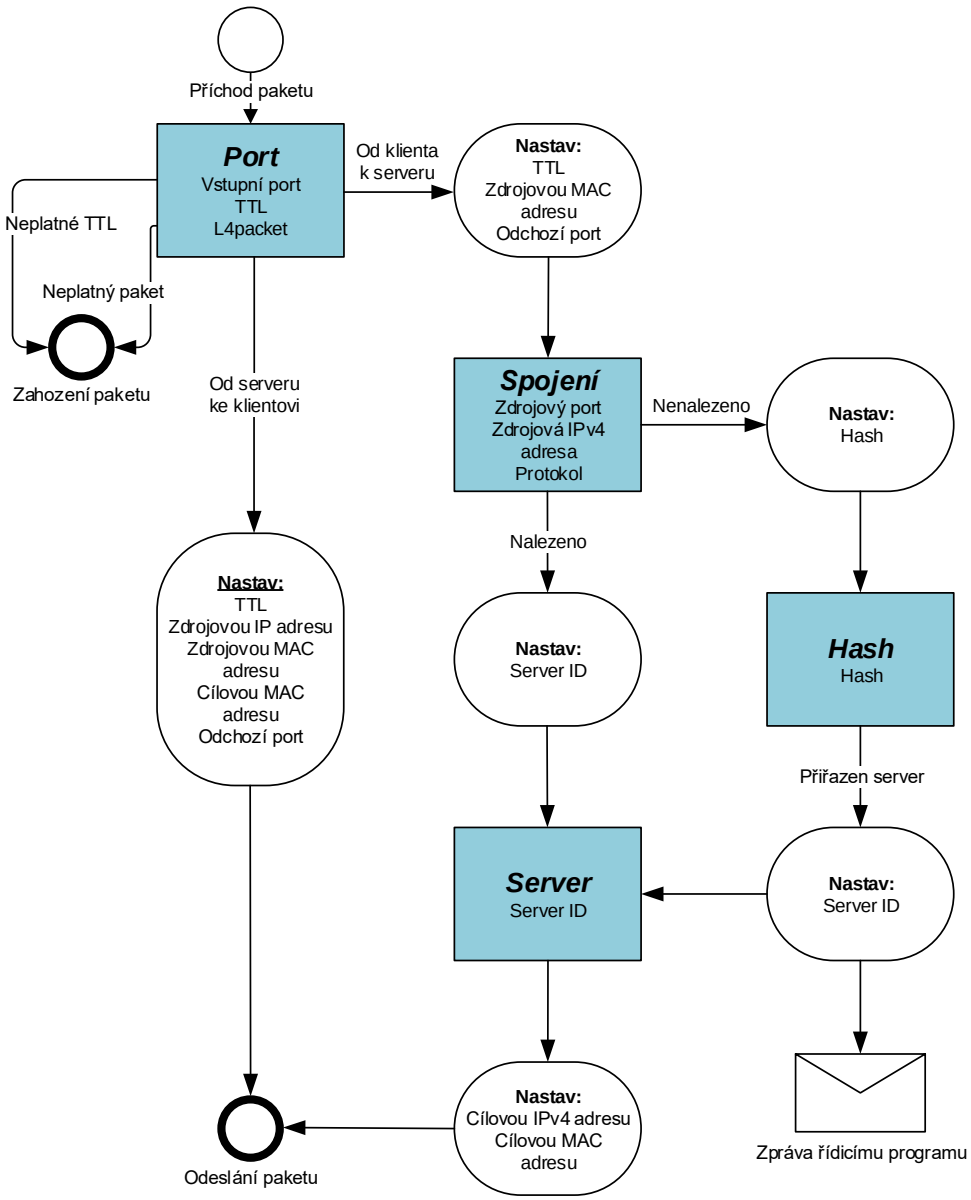
Protože v jazyce $P4_{14}$ nelze volat akce přímo z kontrolní části P4 aplikace, je nutné jakoukoli akci volat přes tabulku. Tabulka *port* proto sdružuje několik různých akcí, které je potřeba vykonat v určitých případech. Tabulka se stará o:

- zahození paketu z důvodu nevalidního TTL,
- zahození paketu z důvodu nepodporovaného protokolu,
- přeposlání paketu od serveru ke klientovi,
- zahájení procesu zpracování paketu putujícího od klienta k serveru.

V tabulce se vyhledává pomocí následujících parametrů:

- vstupní rozhraní,
- TTL,
- informace o použití podporovaných protokolů.

U parametru TTL navíc není vyžadována shoda, ale příslušnost k rozsahu hodnot uvedeného u pravidla. Pravidla jsou běžně za provozu v této tabulce jen dvě. Jedno pro pakety od klienta k serveru, druhé pro opačný směr. Každé z pravidel zároveň vyžaduje validní TTL a použití podporovaných protokolů. Tabulka má proto nastavenou výchozí akci na zahození paketu. Pokud tedy dorazí paket, kterému již vypršelo TTL, nebo pokud paket nepoužívá podporované protokoly, použije se výchozí akce a paket se zahodí.



Obrázek 2.2: Schéma algoritmu P4 aplikace

Pokud přijde paket, který jde od serveru ke klientovi, provede se nastavení zdrojové IPv4 adresy, zdrojové MAC adresy, cílové MAC adresy a odchozího portu a upraví se hodnota TTL.

Paketu, který směřuje od klienta k serveru, se nastaví zdrojová MAC adresa a odchozí port a upraví se hodnota TTL, neboť už víme, že paket půjde pomocí vnitřního rozhraní směrem k serveru. Nevíme však, na který server má být odeslán, proto pokračuje zpracování paketu v tabulce *spojení*.

2.1.6.2 Spojení

Tabulka *spojení* obsahuje informace o spojeních, která už jsou známá a je tedy možné je rovnou přeposlat na cílový server. V tabulce se pro vyhledání pravidla používají následující položky:

- zdrojová IPv4 adresa,
- zdrojový port,
- použitý protokol.

Pravidla v tabulce neobsahují konkrétní informace o cílovém serveru, ale pouze jeho jednoznačný identifikátor, neboť spojení bude pravděpodobně velké množství a nemělo by smysl mít informace o serveru mnohokrát duplikované.

V případě, že právě zpracovávaný paket patří do spojení, které je již známé, nastaví si do interní proměnné informaci o identifikátoru serveru a pokračuje tabulkou *server*.

Pokud však spojení ještě známé není, musí se vybrat server, který ho obslouží. Dojde tedy k vyvolání výchozí akce tabulky, která vypočítá pseudonáhodné číslo, a pokračuje se tabulkou *hash*.

Softwarový nástroj bohužel aktuálně nepodporuje stárnutí pravidel, ačkoli jazyk $P4_{14}$ jako takový má stárnutí pravidel ve specifikaci. Jediný způsob, jak pravidla po určité době z tabulky *spojení* mazat, je pomocí řídícího programu. Tím by se výrazně zvýšila komunikace přes konfigurační kanál z řídicího programu, což by mělo neblahý vliv na výkonnost celého řešení. V práci proto nejsou pravidla z tabulky mazána jednotlivě, ale v případě naplnění tabulky dá řídicí program povel ke smazání všech záznamů. Tím nedochází k nadměrnému zatěžování komunikačního kanálu mazáním jednotlivých pravidel a zároveň je umožněn kontinuální běh aplikace.

2.1.6.3 Hash

Tabulka *hash* obsahuje fixní počet pravidel, která obsahují identifikátory serverů, jež mají aktuálně nejvíce volných prostředků. Poměr zastoupení jednotlivých serverů v této tabulce respektuje poměr volných kapacit serverů, tedy čím více volných prostředků server má, tím více pravidel v této tabulce získá.

V této tabulce se vyhledává pouze pomocí již vypočítaného pseudonáhodného čísla a vždy se najde odpovídající pravidlo obsahující identifikátor serveru, který bude spojení obsluhovat. Identifikátor obsluhujícího serveru se nastaví do interní proměnné a pokračuje se tabulkou *server*. Protože však proběhlo přiřazení nového spojení k serveru, je odeslána zpráva řídicímu programu, který zajistí přidání pravidla do tabulky *spojení*.

2.1.6.4 Server

Tabulka *server* pouze na základě nastaveného identifikátoru obsluhujícího serveru vydá informace o cílové IPv4 a MAC adrese, které následně spuštěná akce nastaví paketu. Ten poté může být odeslán k serveru.

2.2 Kontrolní programy

Kontrolní programy ovlivňují chování loadbalanceru pomocí zásahů do jeho tabulek. Vzhledem k potřebě napojení na stávající nástroje jsou kontrolní programy implementovány v jazyce Python 2.7 [15].

2.2.1 Vyvažování zátěže

Kontrolní program na vyvažování zátěže ovládá tabulku *hash* v $P4_{14}$ aplikaci. Tato tabulka obsahuje fixní počet pravidel a jsou v ní dle poměrů volných kapacit zastoupeny jednotlivé servery. Tento program podle informací o volných kapacitách jednotlivých serverů upravuje pravidla tabulky *hash*. Rozložení tabulky *hash* podle aktuálně volných kapacit serverů znázorňuje obrázek 1.8. V práci se informace o volných kapacitách načítají ze souboru, ale v budoucnosti je možné funkci vyměnit a načítat informace z jiných zdrojů. Ke komunikaci s $P4_{14}$ aplikací používá program rozhraní runtime CLI, které je do programu importováno jako modul, a volá jeho konkrétní funkce pro odesílání zpráv $P4_{14}$ aplikaci. Aktualizace tabulky *hash* je prováděna v pravidelných intervalech.

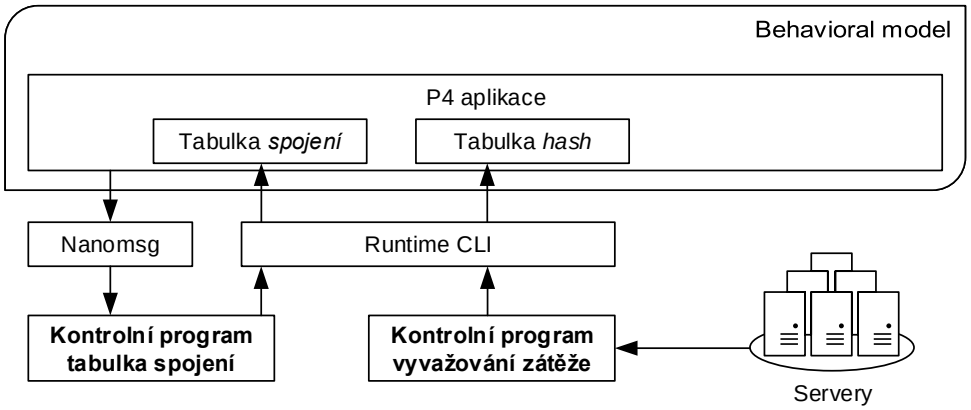
Program zajišťuje také úvodní nastavení $P4_{14}$ aplikace. Jedná se o nastavení:

- výchozích akcí tabulek,
- vnějšího a vnitřního rozhraní,
- informací o obsluhujících serverech.

2.2.2 Tabulka spojení

Druhý kontrolní program má na starosti plnění dat do tabulky *spojení*. V případě, že $P4_{14}$ aplikace určí obsluhující server pro nové spojení, obdrží řídicí

program pomocí protokolu nanomsg zprávu obsahující zdrojovou IPv4 adresu, zdrojový port a protokol. Na základě této zprávy přidá do tabulky *spojení* odpovídající pravidlo. Program má importován jako modul rozhraní runtime CLI a nanomsg klient. Pomocí funkcí runtime CLI zasahuje do tabulek $P4_{14}$ aplikace a pomocí funkcí nanomsg klienta přijímá a potvrzuje zprávy od $P4_{14}$ aplikace. Schéma komunikace obou řídicích programů ilustruje obrázek 2.3.



Obrázek 2.3: Komunikace kontrolních programů s P4 aplikací

2.3 Testování

P4 aplikace i kontrolní programy byly testovány na funkčnost a výkonnost. Otestován byl výkon P4 aplikace v softwarovém běhovém prostředí a schopnost rozdělovat požadavky dle volné kapacity serverů. V této sekci jsou také popsány programy použité při testování a ladění P4 aplikace.

2.3.1 Scapy

Scapy [16] je knihovna v jazyce Python, která umožňuje odesílat pakety se specifikovanými hlavičkami na vybrané rozhraní a kterou lze ovládat přes příkazovou řádku. Knihovna Scapy 2.2.0 byla použita pro odesílání testovacích paketů při ladění P4 aplikace a jako modul testovacího programu při testování rozkládání zátěže.

2.3.2 Wireshark

Program Wireshark [17] umožňuje zachytávat a analyzovat síťový provoz. Poskytuje detailní rozbor zachyceného síťového provozu, včetně například kontroly kontrolního součtu zachyceného paketu. Wireshark 2.2.6 byl použit při ladění P4 aplikace a při testování výkonu.

2.3.3 Testování výkonu

Testování výkonu si kladlo za cíl otestovat výkon P4 aplikace v softwarovém běhovém prostředí pro P4 aplikace. Testování probíhalo na třech počítačích ve virtuálním prostředí pomocí nástroje *Oracle VM VirtualBox 5.1.30* [18]. Jeden zastával funkci webových serverů, druhý představoval P4 aplikaci a poslední simuloval klienty a jejich požadavky. Každý z virtuálních počítačů měl k dispozici 4 GB RAM a dva procesory Intel Core i7-6700HQ s taktem 3,5 GHz. Jako operační systém byl použit Ubuntu 14.04 z důvodu jeho podpory běhovým prostředím pro P4 aplikace.

Při tomto testu se z klientského počítače odesílaly HTTP požadavky přes P4 aplikaci na počítač, kde běžely webové servery. Jako webový server byl použit modul *SimpleHTTPServer 0.3* [19] jazyka Python, pro vytváření HTTP požadavků potom program *wget 1.17.1* [20] a pro sledování vytížení CPU byla použita sada nástrojů *sysstat 11.2.0* [21]. Webové servery běžely dva na jednom počítači a jejich poměr vyvažování byl nastaven na 1:1. Testovány byly různé velikosti načítaných stránek a různá frekvence zasílání požadavků. Sledovány pak byly:

1. doba zpracování HTTP požadavků,
2. využití CPU procesem P4 aplikace,
3. využití paměti RAM procesem P4 aplikace,
4. počet přenesených paketů.

Celkem byly provedeny tři sady testů, každá sada vždy po čtyřech testech. Jednotlivé sady se mezi sebou lišily velikostí načítané stránky a počtem HTTP požadavků, jednotlivé testy v sadě pak dělil interval mezi jednotlivými HTTP požadavky. Naměřené hodnoty z jednotlivých sad jsou uvedeny v tabulkách 2.1, 2.2 a 2.3.

Tabulka 2.1: Testování výkonu, 10000 HTTP požadavků na 9 B stránku

Test	**1**	**2**	**3**	**4**
Celkový čas testu	58,7 s	82,3 s	157,4 s	257,2 s
Interval mezi požadavky	0 ms	2,5 ms	10 ms	20 ms
Celkem přeneseno paketů	100040	100022	100014	100012
Chybné pakety	40	22	14	12
Požadavků za sekundu	170,4	121,5	63,5	38,9
Čas na požadavek	5,9 ms	5,7 ms	5,7 ms	5,7 ms
Paketů na požadavek	10,0	10,0	10,0	10,0
Vytížení CPU procesem	47,5 %	34,9 %	19,3 %	12,3 %

Doba zpracování se zvyšuje společně s rostoucím intervalem mezi požadavky. V tomto ohledu je zajímavější sledovat průměrný čas potřebný pro

Tabulka 2.2: Testování výkonu, 1000 HTTP požadavků na 100 kB stránku

Test	**1**	**2**	**3**	**4**
Celkový čas testu	46,2 s	55,1 s	65,3 s	95,5 s
Interval mezi požadavky	0 ms	10 ms	20 ms	50 ms
Požadavků za sekundu	21,6	18,1	15,3	10,5
Čas na požadavek	46,2 ms	45,1 ms	45,3 ms	45,5 ms
Vytížení CPU procesem	86,7 %	68,8 %	59,3 %	41,3 %

Tabulka 2.3: Testování výkonu, 100 HTTP požadavků na 1 MB stránku

Test	**1**	**2**	**3**	**4**
Celkový čas testu	22,7 s	24,1 s	27,2 s	32,1 s
Interval mezi požadavky	0 ms	20 ms	50 ms	100 ms
Požadavků za sekundu	4,4	4,1	3,7	3,1
Čas na požadavek	227,0 ms	221,0 ms	222,0 ms	221,0 ms
Vytížení CPU procesem	97,5 %	92,2 %	87,3 %	82,8 %

obsloužení jednoho požadavku. Z tabulek je možné vidět, že při testu, kde nebyl mezi požadavky žádný interval, byl čas potřebný k obsloužení každého požadavku mírně vyšší. Jakmile byl mezi jednotlivými požadavky alespoň nějaký interval, zůstávala doba zpracování jednotlivých požadavků téměř stejná.

Vytížení CPU bylo nejvyšší, když mezi jednotlivými HTTP požadavky nebyl žádný interval, a snižovalo se společně s rostoucím intervalem mezi jednotlivými požadavky. Když rostl interval mezi požadavky, snižoval se počet požadavků za sekundu a tím se snižovala zátěž na CPU. Zároveň bylo využití CPU nižší při načítání menších stránek. To lze vysvětlit tím, že při větších stránkách bylo potřeba z důvodu fragmentace více TCP fragmentů, které musely projít skrz P4 aplikaci.

Na využití paměti RAM se zátěž nijak neprojevovala. Proto také není uvedeno využití paměti RAM v tabulkách naměřených hodnot. Aplikace si nejspíše již při startu alokuje místo na tabulky, jejichž maximální velikost zná, a nemusí potom provádět další alokace za běhu. To je výhodné z hlediska výkonu, ale může to znamenat větší spotřebu paměti, než je skutečně třeba, protože se zabírá paměť, do které ještě není potřeba nic ukládat.

U první sady testů s velikostí stránky 9 B byl měřen i počet přenesených paketů a jejich chybovost. U dalších testů už vzhledem k vyššímu datovému toku zachytávání provozu negativně ovlivňovalo výkon P4 aplikace, takže nebylo prováděno. Za chybový paket byl považován paket s odpovědí, která už byla zaslána, nebo opakování již odeslaného datového paketu. Zde je možné sledovat, že s rostoucími intervaly mezi jednotlivými HTTP požadavky se chybovost snižuje. Důvodem je, že P4 aplikace zpracuje za stejný časový interval méně paketů a má tedy více času na případnou reakci.

2.3.4 Testování rozkládání zátěže

Test rozkládání zátěže má za cíl otestovat, jak P4 aplikace zvládá rozdělovat požadavky dle poměru volných kapacit serverů. Při tomto testu byly do P4 aplikace zasílány pakety s pseudonáhodně vygenerovanou zdrojovou IPv4 adresou a zdrojovým TCP portem. Celý test, včetně generování pseudonáhodných čísel, byl realizován pomocí jazyka Python s využitím modulu Scapy. Bylo sledováno, jak P4 aplikace požadavky rozdělí v závislosti na nastaveném poměru volných kapacit serverů. Testy byly provedeny pro různé poměry volných kapacit serverů a různé počty požadavků. Výsledky pro vhodný počet záznamů v tabulce *hash* (40) jsou uspořádány v tabulce 2.4, kde je vidět, že P4 aplikace zvládá vhodně rozdělovat požadavky již od jejich malého počtu.

Tabulka 2.4: Testování rozkládání zátěže, 40 pravidel v tabulce *hash*

Celkem požadavků	**Poměr rozložení**	**Rozdělení požadavků loadbalancerem**			
		Server 0	**Server 1**	**Server 2**	**Server 3**
10	1:1:0:0	4	6	0	0
100	1:1:0:0	47	53	0	0
1000	1:1:0:0	503	497	0	0
10	1:1:1:1	2	4	2	2
100	1:1:1:1	23	25	28	24
1000	1:1:1:1	240	239	250	271
10	1:2:3:4	1	2	2	5
100	1:2:3:4	12	17	33	38
1000	1:2:3:4	94	185	299	422
10	0:1:5:10	0	1	3	6
100	0:1:5:10	0	6	33	61
1000	0:1:5:10	0	70	323	607

Tabulka 2.5: Testování rozkládání zátěže, 100 požadavků

Záznamů v *hash*	**Poměr rozložení**	**Rozdělení požadavků loadbalancerem**			
		Server 0	**Server 1**	**Server 2**	**Server 3**
80	0:1:5:10	0	7	32	61
40	0:1:5:10	0	6	30	64
20	0:1:5:10	0	13	33	54
10	0:1:5:10	0	15	26	59
5	0:1:5:10	0	18	17	65

V tabulce 2.5 jsou potom výsledky rozložení zátěže při odeslání 100 náhodných požadavků a různém počtu záznamů v tabulce *hash*. Je možné vidět, že při nižším počtu záznamů v tabulce *hash* již není rozkládání natolik přesné. To je proto, že zkrátka nelze vhodným způsobem rozdělit 5 záznamů v poměru

0:1:5:10. Pro optimální fungování je vhodné mít v tabulce *hash* minimálně 10 záznamů za každý obsluhovaný server.

2.4 Portace na kartu 100 Gb/s COMBO

Karta 100 Gb/s COMBO je síťová akcelerační karta připojitelná přes sběrnici PCI Express, jejíž jádrem je výkonné programovatelné hradlové pole [22], [23]. Pro tuto kartu existuje překladač (aktuálně ve verzi 1.1.5) z jazyka P4 vyvíjený sdružením CESNET. Jedním z cílů práce je prozkoumat možnost přenositelnosti implementované P4 aplikace na platformu karty 100 Gb/s COMBO.

Bylo zjištěno, že pro portaci na kartu 100 Gb/s COMBO je potřeba rozšířit překladač o podporu `field_list` a `field_list_calculation`, které jsou nutné k výpočtu kontrolních součtů, a také o podporu primitivní akce `modify_field_with_hash_based_offset` na náhodný výběr čísla z určitého intervalu, která je používána k výběru obsluhujícího serveru. Dále je potřeba doplnit podporu stárnutí pravidel v tabulkách, bez které lze jen velmi obtížně mazat z tabulek stará pravidla. Podpora stárnutí pravidel dosud není v P4 překladači pro karty implementována, protože sdružení CESNET zatím nemělo na tuto funkcionalitu požadavek. Bez implementované podpory stárnutí pravidel by bylo možné nepotřebná pravidla mazat pomocí řídicího softwaru, což by však znamenalo snížení propustnosti z důvodu vyšší komunikace přes řídicí sběrnici. Zajímavá by jistě byla i taková implementace stárnutí pravidel, která by zohledňovala nikoli čas od zavedení pravidla do tabulky, ale čas od poslední aplikace daného pravidla. To by umožňovalo udržovat v tabulkách pouze ta pravidla, která jsou aktuálně používána. Identifikované problémy byly nahlášeny autorům, kteří se na základě implementované P4 aplikace rozhodli rozšířit funkcionalitu překladače.

Po rozšíření překladače bude P4 aplikace přenositelná a rozšiřitelná na kartu 100 Gb/s COMBO. Pro samotnou portaci se však bude muset přepsat i kontrolní aplikace, protože celá infrastruktura nástrojů a ovladače nepodporuje konfiguraci přes IPC rozhraní a přijímání zpráv přes nanomsg. Pro konfiguraci pravidel a ovládání lze použít knihovnu *libp4dev*, která umožňuje přímou manipulaci s vygenerovaným P4 zařízením. Přijímání učících zpráv je možné realizovat pomocí napojení na DMA, který umožňuje zasílat zprávy z HW přes SZE kanál [24]. Aplikaci je možné připojit k DMA kanálu pomocí knihovny *libsze2*. Všechny tyto knihovny a nástroje byly vytvořeny sdružením CESNET a nejsou veřejně dostupné. Přístup k nim byl umožněn na základě navázání spolupráce s projektem Liberouter [25].

Závěr

Cílem práce bylo navrhnout, implementovat a otestovat aplikaci v jazyce P4, která slouží k vyvažování zátěže webového provozu, a diskutovat možnost portace vzniklé P4 aplikace na kartu 100 Gb/s COMBO.

V kapitole Analýza a návrh byly představeny známé přístupy k vyvažovaní zátěže webového provozu, byl popsán jazyk P4 spolu s nástroji potřebnými pro vývoj a testování a byla navržena vhodná P4 aplikace na vyvažování zátěže. V kapitole Implementace byla implementována navržená P4 aplikace včetně řídicích programů, pomocí testování byla úspěšně ověřena funkčnost a změřena výkonnost P4 aplikace a závěrem byla diskutována možnost portace P4 aplikace na kartu 100 Gb/s COMBO. Tím práce splnila všechny body zadání. Výsledná P4 aplikace může dokonce díky nízké úrovni, na které rozděluje zátěž, fungovat i pro jiné služby než jen pro webový provoz.

Testováním bylo zjištěno, že P4 aplikace nasazená v běhovém prostředí zvládne obsloužit za sekundu až 170 HTTP požadavků s velmi malou odpovědí a nejvíce ji vytíží HTTP požadavky s velkou odpovědí, u kterých je potřeba zpracovat velké množství TCP fragmentů. Dále bylo také ověřeno, že aplikace zvládne vhodně rozdělovat požadavky dle vytížení jednotlivých serverů.

Na základě práce bylo inicializováno další rozšíření překladače pro kartu 100 Gb/s COMBO, po kterém by bylo možné provést portaci P4 aplikace na kartu. Po vytvoření řídicích programů by pak bylo možné provozovat P4 aplikaci s výkonem akceleračních karet vybavených programovatelnými hradlovými poly. Práci je také možné do budoucnosti rozšířit o podporu dalších síťových protokolů, například IPv6.

Během práce jsem se potýkal s několika problémy způsobenými tím, že běhové prostředí i jeho překladač jsou stále ve vývoji a některé jejich funkcionality tak obsahují chyby, nebo ještě nejsou implementovány.

Literatura

[1] HARDESTY, L.: *Google Brings SDN to the Public Internet [online].* SDxCentral, duben 2017, [cit. 2017-12-17]. Dostupné z: `https://www.sdxcentral.com/articles/news/google-brings-sdn-public-internet/2017/04/`

[2] BRISCO, T.: *RFC 1794 - DNS Support for Load Balancing [online].* The Internet Engineering Task Force, duben 1995, [cit. 2017-12-02]. Dostupné z: `http://www.ietf.org/rfc/rfc1794.txt`

[3] INTERNATIONAL TELECOMMUNICATION UNION: *Recommendation X.200 [online].* International Telecommunication Union, červenec 1994, [cit. 2017-12-16]. Dostupné z: `https://www.itu.int/rec/T-REC-X.200-199407-I/en`

[4] LAW, D. J.; DIAB, W. W.; HEALEY, A.; aj.: *IEEE Std 802.3^{TM}-2012 - IEEE Standard for Ethernet [online].* Institute of Electrical and Electronics Engineers, prosinec 2012, [cit. 2017-12-02]. Dostupné z: `http://standards.ieee.org/getieee802/download/802.3-2012_section1.pdf`

[5] INFORMATION SCIENCES INSTITUTE: *Internet Protocol [online].* The Internet Engineering Task Force, září 1981, [cit. 2017-12-02]. Dostupné z: `http://www.ietf.org/rfc/rfc791.txt`

[6] DEERING, S. E.; HINDEN, R. M.: *Internet Protocol, Version 6 (IPv6) Specification [online].* The Internet Engineering Task Force, červenec 2017, [cit. 2017-12-18]. Dostupné z: `http://www.ietf.org/rfc/rfc8200.txt`

[7] INFORMATION SCIENCES INSTITUTE: *Transmission Control Protocol [online].* The Internet Engineering Task Force, září 1981, [cit. 2017-12-02]. Dostupné z: `http://www.ietf.org/rfc/rfc793.txt`

[8] INFORMATION SCIENCES INSTITUTE: *User Datagram Protocol [online]*. The Internet Engineering Task Force, srpen 1980, [cit. 2017-12-02]. Dostupné z: http://www.ietf.org/rfc/rfc768.txt

[9] BENÁČEK, P.; PUŠ, V.: *Jazyk P4 jako budoucnost SDN [online]*. Root.cz, leden 2016, [cit. 2017-12-03]. Dostupné z: https://www.root.cz/clanky/jazyk-p4-jako-budoucnost-sdn

[10] P4 COMMUNITY: *P4 Language Consorcium [online]*. prosinec 2017, [cit. 2017-12-18]. Dostupné z: https://p4.org

[11] THE P4 LANGUAGE CONSORTIUM: *The P4 Language Specification Version 1.0.4 [online]*. The P4 Language Consortium, květen 2017, [cit. 2017-12-08]. Dostupné z: https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf

[12] BENÁČEK, P.; PUŠ, V.: *Jazyk P4 jako budoucnost SDN (dokončení) [online]*. Root.cz, leden 2016, [cit. 2017-12-03]. Dostupné z: https://www.root.cz/clanky/jazyk-p4-jako-budoucnost-sdn-dokonceni

[13] P4 COMMUNITY: *Behavioral-model [online]*. červen 2017, [cit. 2017-12-10]. Dostupné z: https://github.com/p4lang/behavioral-model

[14] P4 COMMUNITY: *P4C [online]*. prosinec 2017, [cit. 2017-12-09]. Dostupné z: https://github.com/p4lang/p4c

[15] Python Software Foundation: *Python [online]*. prosinec 2017, [cit. 2017-12-18]. Dostupné z: https://www.python.org

[16] BIONDI, P.; VALADON, G.; LALET, P.: *Scapy [online]*. prosinec 2017, [cit. 2017-12-18]. Dostupné z: http://www.secdev.org/projects/scapy

[17] COMBS, G.; RAMIREZ, G.; BOTTOM, T.; aj.: *Wireshark [online]*. listopad 2017, [cit. 2017-12-16]. Dostupné z: https://www.wireshark.org

[18] Corporation, O.: *Oracle VM VirtualBox [online]*. prosinec 2017, [cit. 2017-12-19]. Dostupné z: https://www.virtualbox.org

[19] Python Software Foundation: *Simple HTTP request handler [online]*. prosinec 2017, [cit. 2017-12-18]. Dostupné z: https://docs.python.org/2/library/simplehttpserver.html

[20] NIKŠIĆ, H.; RÜHSEN, T.; SHAH, D.; aj.: *GNU Wget [online]*. září 2017, [cit. 2017-12-18]. Dostupné z: https://www.gnu.org/software/wget

[21] GODARD, S.: *sysstat [online]*. listopad 2017, [cit. 2017-12-18]. Dostupné z: http://sebastien.godard.pagesperso-orange.fr

[22] CESNET, S.: *COMBO-100G2 [online]*. prosinec 2017, [cit. 2017-12-20]. Dostupné z: `https://www.liberouter.org/combo-100g2`

[23] CESNET, S.: *COMBO-100G2Q [online]*. prosinec 2017, [cit. 2017-12-20]. Dostupné z: `https://www.liberouter.org/combo-100g2q`

[24] CESNET, S.: *NetCOPE [online]*. prosinec 2017, [cit. 2017-12-21]. Dostupné z: `https://www.liberouter.org/technologies/netcope`

[25] CESNET, S.: *Liberouter [online]*. prosinec 2017, [cit. 2017-12-21]. Dostupné z: `https://www.liberouter.org/technologies/netcope`

Příloha A

Seznam použitých zkratek

API Application Programming Interface

ARP Address Resolution Protocol

CPU Central Processing Unit

DNS Domain Name System

HTTP Hypertext Transfer Protocol

IEEE Institute of Electrical and Electronics Engineers

IP Internet Protocol

IPC Inter-Process Communication

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

MAC Media Access Control

P4 Programming Protocol-Independent Packet Processors

PCI Peripheral Component Interconnect

RAM Random Access Memory

SDN Software-Defined Networking

TCP Transmission Control Protocol

TTL Time To Live

UDP User Datagram Protocol

Příloha B

Spuštění simulace

Pro zprovoznění P4 aplikace v běhovém prostředí je třeba mít funkční nástroj behavioral model [13] a komunitní P4 překladač [14]. Řídicí programy `controlhash.py`, `controlconnection.py`, knihovnu `message_client.py` a soubor `server_stats` je třeba nakopírovat do složky `tools` v nástroji behavioral model. V této složce je umístěn skript, kterým je možné vytvořit virtuální rozhraní pro připojení P4 aplikace. Překlad implementace je možné provést pomocí:

```
P4_TRANSLATOR/p4c -x p4-14 -b bmv2-ss-p4org
    --p4runtime-format json INPUT.P4 -o /OUTPUT_FOLDER
```

Překladem se získá popis chování P4 aplikace v jazyce JSON. Tento popis je možné simulovat prostřednictvím zařízení `simple_switch` v nástroji behavioral model:

```
BEHAVIORAL_MODEL/targets/simple_switch/simple_switch
    -i 0@INTERFACE_OUTSIDE -i 1@INTERFACE_INSIDE
    --nanolog ipc:///tmp/log.ipc
    --notifications-addr ipc:///tmp/notification.ipc
    INPUT.JSON
```

Po spuštění simulace P4 aplikace je ve složce `tools` v nástroji behavioral model možné spustit řídicí programy `controlhash.py` a `controlconnection.py`. V programu `controlhash.py` je také možné upravit proměnné, které rozhodují o nastavení P4 aplikace.

Pro správné fungování je potřeba, aby byly na rozhraních, ke kterým je P4 aplikace připojena, nastaveny odpovídající IPv4 a MAC adresy a v ARP tabulkách byly zavedeny záznamy pro rozhraní P4 aplikace. Rozhraním, které budou odesílat odpovědi klientům, je pak potřeba nastavit jako výchozí bránu vnitřní rozhraní P4 aplikace. K práci jsou přiloženy také testy `testspeed.py` a `testloadbalancing.py`, kterými byla P4 aplikace testována.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD a spuštění simulace
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `other` ..... ostatní soubory potřebné ke spuštění práce
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `pictures` ..... zdrojové formy všech použitých obrázků
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF