VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

http://fit.vut.cz

Projektová praxe 2019/2020

Ovladač libp4dev verze 4

Radek Veverka*

Abstrakt

Cílem práce bylo seznámit se se současnou implementací ovladače P4 zařízení, zvaného libp4dev, jenž je součástí P4 ekosystému vyvíjenému CESNETem, a následně implementovat novou verzi, která bude v souladu s novými požadavky specifikovanými sdružením CESNET a firmou Netcope. Práce se věnuje počátkům implementace této knihovny, konkrétně modulu pro zpracování device tree.

Klíčová slova: libp4dev — ovladač — P4

Přiložené materiály: *N/A*

*xvever13@stud.fit.vut.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

Projekt CESNETu NFV200 se věnuje platformě pro virtualizaci síťových funkcí v prostředí nejrychlejších sítí, základem byla síťová karta se dvěma 100G ethernetovými rozhraními [1]. V rámci tohoto projektu vzniká celá řada nástrojů, mimo jiné mezi ně patří i nástroje pro podporu vývoje síťových aplikací v jazyce P4. Pomocí tohoto jazykya lze efektivně definovat, jakým způsobem budou zpracovány pakety proudící skrze síťové zařízení. Jazyk P4 je však obecný a není svázan s žádným konkrétním typem zařízení. Z toho důvodu je překladač P4 jazyka rozdělen na společnou část a specifickou část. Společná část (frontend) provádí lexikální, syntaktickou a části sémantické analýzy, zatímco specifická (backend) překládá vnitřní reprezentaci programu na cílový kód dle toho, jak je naprogramováná uživatelem překladače. CESNET vyvinul překladač, který transformuje kód do VHDL, který je následně nahrán na čip FPGA na vysokorychlostní kartě. Z velmi obecného pohledu se architektura běžného síťového zařízení (např. routeru) dělí na control plane a data plane. [2] Data plane je část, kterou prochází všechny pakety, pro dosažení vysokých rychlostí musí být tedy hardwarově akcelerovaná. Data plane extrahuje hodnoty v hlavičkách paketu, podle kterých se vyhledává v match+action tabulkách příslušná akce. Tabulky jsou také hardwarově akcelerované. Control plane je softwarová část zařízení, která slouží mimo jiné také k plnění tabulek pravidly. Funguje jako řídící jednotka, která určuje, jak se k paketům chová data plane. Pokud data plane neví, co s paketem provést, požádá o informaci control plane. Běžný provoz však přes control plane neproudí, protože by to výrazně omezovalo rychlost zařízení. V P4 se programuje právě část data plane, tedy extrakce hlaviček, definice match+action tabulek a kontrolního programu.

Kromě překladače je tedy nutná existence dalšího nástroje, který bude umět spravovat pravidla v tabulkách a obsah registrů a čítačů. Právě tyto funkce má na starosti knihovna libp4dev.

2. Ovladač libp4dev verze 3

Současná verze ovladače je implementována v jazyce C. V prvotních verzích byl kladen důraz především na rychlost implementace a hlavně jednoduchost používání. Počítalo se s tím, že program dobře poslouží pouze jednoduchému účelu, a to k připojení na kartu a naplnění tabulek pravidly. Nicméně později se ukázalo, požadavky na nástroj přibývají. Byly doimplementovány registry a čítače. Výsledkem byl návrhově poměrně komplikovaný software, který navíc nedodržoval základní principy objektově orientovaného přístupu. Proto se spolupracující společnosti CESNET a Netcope dohodly, že bude dobré reimplementovat ovladač s do-

držením předem stanoveného návrhu.

3. Požadavky na nový ovladač libp4dev verze 4

Ovladač bude vytvořen jako knihovna v jazyce C. Je to proto, aby jej bylo možná přikompilovat jako modul jádra Linuxu. Toto zůstává stejné jako ve verzi 3. Z tohoto důvodu je také nutné důsledně pracovat s paměťovými zdroji a korektně je uvolňovat.

Ovladač bude objektově orientován. I přesto že jazyk C je strukturovaný a nepodporuje přímo objektově orientované programování, je možné v něm dodržovat zásady objektově orientovaného přístupu. Místo tříd poslouží struktury, místo metod obyčejné funkce. Jazyk C je modulární, takže funkce a struktury se rozdělí do více samostatných překladových jednotek. Každý dynamicky alokovaný objekt (struktura), by měl mít inicializační funkci, která se postará o korektní přidělení paměti a inicializaci položek na výchozí hodnoty. Pokud existuje inicializační funkce, musí existovat také funkce s opačným úkolem, tedy dealokovat vše, co alokovala inicializační funkce, ne však nic navíc.

Pro každou entitu relevantní pro ovladač, například tabulku, zařízení nebo pravidlo, bude existovat odpovídající objekt (struktura). Tento přístup nebyl zcela dodržen ve verzi 3, kde třeba tabulka, což je nejpodstatnější entita ovladače, neměla vlastní objektovou reprezentaci. Toto vedlo k znesnadnění pochopení a práce s kódem knihovny.

Pro každou entitu budou existovat jednotkové testy. Z toho důvodů je nutné co nejvíce redukovat počet závislostí mezi entitami a především se vyvarovat cyklickým závislostem.

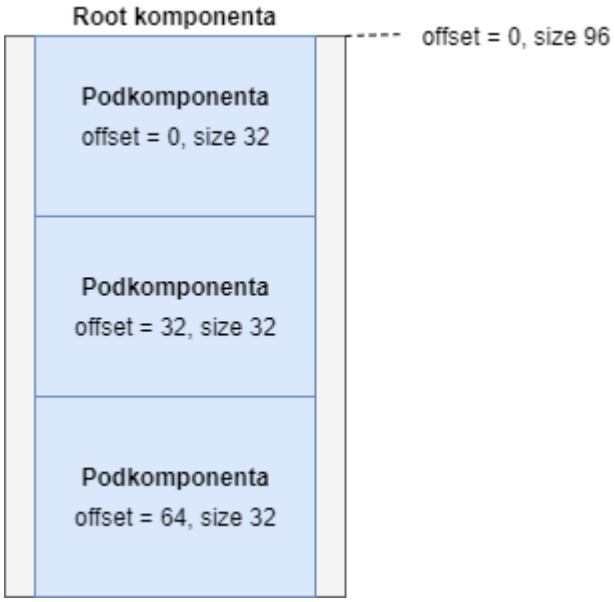
Způsob zápisu a čtení dat ze zařízení bude z knihovny vyčleněn a bude v kompetenci uživatele knihovny. Obecně zápis a čtení z cílových zařízení probíhá tak, že se zavolá nějaká knihovní funkce pro dané zařízení, která umožňuje přečíst nebo zapsat specifický počet bajtů s počátečními offsetem (místem zápisu prvního bajtu). Problém však je, že různé karty poskytují různé knihovní funkce a mají různé offsety, kde se nachází P4 pipeline. Ve verzi 3 se toto řešilo nepříliš čistými technikami, například s využitím podmíněného překladu preprocesoru. Pro zachování univerzálnosti knihovny a možnost použití pro více zařízení je tedy nutné, aby uživatel předal potřebné informace, tedy kam a jak se bude zapisovat. Pro tento účel se využije malá knihovna libmi32.

Data z device tree by měla být zpracována a uložena do struktur odděleně od zbytku ovladače. V těchto strukturách by neměly být žádné vazby na zbytek ovladače, naopak vazby na tyto struktury ze zbytku ovladače existovat musí. V libp4dev 3 nebyly informace z device tree vůbec zpracovány a uloženy. Device tree tedy bylo prohledáváno a zpracováváno zbytečně opakovaně vždy, když byla potřeba nějaká informace, kód zpracování device tree se navíc nacházel přímo v logice aplikace.

Chyby budou kontrolovány a zpracovávány konzistetním způsobem. Bude existovat seznam vhodně pojmenovaných celočíselných konstant identifikující jednotlivé typy chyb. Každá operace (funkce), která může skončit chybou, bude mít celočíselný návratový typ. Každou možnost chyby řádně zkontroluje, po detekci chyby dealokuje případnou naalokovanou paměť a návratovou hodnotou propaguje chybový kód volající funkci. Tato funkce musí být opět připravena zareagovat uklizením paměti a předáním návratové hodnoty další volající funkcí. Toto se opakuje, dokud se návratová hodnota nevrátí uživateli, který použil některou z API funkcí knihoven. Knihovna musí poskytnout uživateli funkci k zisku statického řetězce, který přísluší danému chybovému kódu a obsahuje název chyby. V případě, že funkce potřebuje navrátit jinou relevantní hodnotu, uloží ji do ukazatele v parametru. Po bezchybovém průběhu funkce je navrácena vždy hodnota 0 (P4DEV_OK), jinak číslo větší než 0, dle typu chyby.

4. libmi32

Knihovna libmi32 je velmi jednoduchá knihovna od Netcope, která bude využívána jak v libp4dev 4, tak v uživatelské aplikaci používající libp4dev 4. Knihovna umožňuje vytvářet dva typy komponent - root component a child component. Komponenta je v podstatě část adresního prostoru, která je dána offsetem a velikostí. Root komponenta reprezentuje celý definovaný adresní prostor. Při jejím vytváření se také předává dvojice ukazatelů na funkce mi32_read a mi32_write, které slouží ke čtení/zápisu 32 bitové hodnoty na specifikovaný offset. Tyto funkce jsou definovány uživatelem před samotným vytvořením root komponenty. Vytvořenou root komponentu (ale i kteroukoliv jinou komponenetu) je možné rozdělit na menší komponenty voláním mi32_create_child a předáním rodičovské komponenty, relativního offsetu a velikosti. Při práci s komponentou pak není nutné ustavičně počítat absolutní offset, za to je totiž zodpovědná daná komponenta ve struktuře ostatních komponent. Zároveň to umožňuje knihovně libp4dev číst a zapisovat transparetně právě pomocí těchto komponent.



Obrázek 1. Paměťový prostor komponent libmi32

5. P4 pipeline a device tree

Aby bylo možné pracovat s P4 zařízením, musí ovladač vědět, co toto zařízení obsahuje. Jinými slovy je třeba, aby překladač znal strukturu P4 pipeline (například informace o tabulkách) načtenou v cílovém zařízení. Tato struktura je definována právě v jazyce P4. Kromě výsledných VHDL a HLS zdrojových kódu pro syntézu vygeneruje backend překladače i popis, které tabulky P4 pipeline obsahuje, jak se jmenují, jaký mají vyhledávací klíč a vyhledávací engine, atd. Tyto informace jsou reprezentovány ve formátu device tree, což je formát pro popis hardwaru. Device tree popis může existovat jak v textové, tak v binární formě.

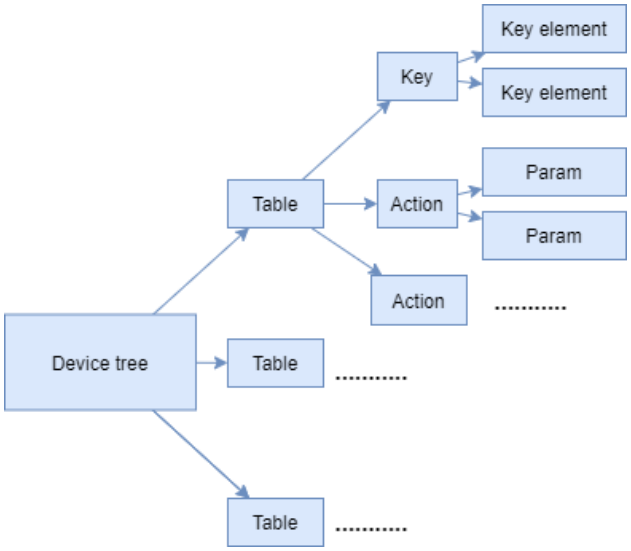
1. **DTS** - device tree source, je textová forma device tree čitelná pro člověka. Právě v této formě generuje backend překladače P4 popis P4 pipeline.
2. **DTB** - device tree blob je binární forma device tree. V této formě pracují s device tree ostatní programy.
3. **DTC** - device tree compiler. Je utilita sloužící k převodu DTS do DTB [3].

Ovladač libp4dev pracuje s DTB s využitím knihovny libfdt.

6. Device tree parser

Device tree parser je modul ovladače, který je schopen zpracovat DTB a naalokovat hierarchii objektů, které odpovídají této struktuře z DTB. Výsledná struktura je samostatná, tedy nezávislá na zbytku ovladače. Tento modul je volán v rámci inicializace zařízení. Mezi objekty (struktury) tohoto modulu patří:

1. `device_tree_t` - Je kořenem celé hierarchie, inicializován pro jedno DTB pouze jednou. Obsahuje offset, velikost, počet tabulek a pole ukazatelů na tabulky. Může také obsahovat podpůrné informace, např. verze device tree nebo P4 překladače. Při inicializai tohoto objektu se nejdříve nalezne uzel v device tree, který odpovídá P4 pipeline. Tím se zároveň odhalí případně nevalidní soubor DTB. V P4 uzlu všechny přímé poduzly reprezentují jednotlivé tabulky, jak jsou seřazeny v pipeline. Proto se následně přes tyto uzly iteruje a volá se zpracování tabulky, kam je předán offset tabulky device tree.
2. `device_tree_table_t` - reprezentuje tabulku v hierarchii device tree. Match+action tabulka je základním stavebním kamenem P4 pipeline. Struktura extrahuje z device tree opět offset a velikost. Navíc extrahuje také jméno tabulky, což je unikátní klíč, podle kterého bude umožněno uživateli knihovny přistupovat ke konkrétní tabulce a pracovat s ní. Nedílnou součástí tabulek je také klíč a akce. Toto však již nejsou atomické položky, proto se z device tree extrahují do samostatných struktur, na které je následně z tabulky odkázáno.
3. `device_tree_key_t` - reprezentuje vyhledávací klíč tabulky. Jelikož tato struktura se vyskytuje v každé tabulce pouze jednou, dává smysl ji zahrnout přímo do struktury tabulky namísto použití ukazatele a dynamické alokace. Vyhledávací klíč si uchovává typ enginu. Tento typ je v device tree reprezentován formou krátkého textového řetězce, který může nabývat jedné z hodnot "lpmbst", "cuckoo", "tcam". Součástí návrhu ovladače je i možnost jednoduše přidat nový engine. Vyhledávací klíč se může skládat z více podklíčů, proto obsahuje dynamicky alokované pole.
4. `device_tree_key_elem_t` - reprezentuje dílčí klíč vyhledávacího klíče tabulky. Tato struktura už nese informaci o tom, které hlavičky se klíč týká, jakou má velikost a jakým způsobem se bude porovnávat. P4.16 specifikaci nabízí porovnání pomocí operátorů lpm (na základě prefixu), exact (přesná hodnota) a ternary (na základě masky). To, který engine tabulky bude zvolen při různých kombinacích typů dílčích klíčů, je v kompetenci překladače P4, který tuto informaci poskytne v device tree.
5. `device_tree_action_t` - Jak bylo zmíněno výše, tabulka obsahuje mimo jiné klíč a seznam akcí. Konfigurace tabulky probíhá tak, že uži-



Obrázek 2. Struktura device tree

vatel zadává pravidla, která mapují konkrétní hodnoty vyhledávacího klíče na konkrétní akci. Akce jsou implementovány v jazyce P4 přeloženy pomocí HLS (high level syntézy). Překladač exportuje do device tree metadata o akcích, tedy jméno, operační kód a seznam parametrů. Parametr opět není atomická hodnota a může jich existovat více, jsou tedy rovněž dynamicky alokovány a odkázány, jako tomu je dílčích klíčů.

6. `device_tree_param_t` - Parametr akce je v podstatě jednoduchý pár hodnot, který se skládá z bitové šířky parametru a řetězcového identifikátoru (jména) parametru.

Ač by každý objekt měl obsahovat konstruktor a destruktor (v C jsou to funkce pro alokaci a dealokaci), v případě triviálních struktur to potřeba není. Jde o struktury, které samy od sebe nealokují žádnou paměť. Inicializační a deinicializační funkce objeků v žádném případě nevytváří ani neruší paměťový prostor, ve kterém se samy nachází. Je to proto, aby uživatel knihovny mohl sám specifikovat, jakým způsobem bude paměť pro objekt přidělena. Objekty spravují pouze tu paměť, kterou alokovaly pro vnitřní potřeby. V předchozích implementacích libp4dev toto chování bylo často matoucí, protože destruktory byly pojmenovány stylem: `free_jmenoentity(ukazatel)`, což naznačovalo, že bude z paměti uvolněn i samotný objekt. Takové chování by ale nebylo přípustné v případě objektů uložených na zásobníku. Proto jsou ve verzi 4 destruktory pojmenovány podle konvence která korektně označuje, co opravdu dělají. Například místo názvu `free_device_tree_table` se použije `free_device_tree_table_content`.

7. Enginy a operace

Různé tabulky mohou být v hardwaru implementovány různými způsoby. Ovladač by proto měl být schopen podporovat více různých implementací (enginů) a snadno je přidávat. Některé operace prováděné ovladačem se mohou lišit v závislosti na tom, jaký engine tabulka používá. Bylo by rozumné vystavit do API společné funkce, pomocí kterých bude uživatel pracovat s knihovnou transparentně, nebude tedy muset řešit který engine je použit u které tabulky. Jde v podstatě o polymorfismus, kde všechny enginy sdílí stejné rozhraní a liší se pouze v implementaci. Otázkou je, jak simulovat polymorfismus v jazyce C.

Nabízí se použití ukazatelů na funkce. Definuje se struktura, která bude obsahovat pojmenované ukazatele na funkce. Pro každý engine se definuje struktura obsahující unikátní identifikátor enginu (například lpmbst) a obecná struktura s funkcemi. Vnitřní struktura enginu (obecná), bude při inicializaci vnější struktury enginu naplněna konkrétními ukazately na funkce z modulu enginu. Všechny funkce modulu musí pochopitelně dodržovat hlavičku předepsanou obecnou strukturou. Dále musí být k dispozici funkce, která na základě identifikátoru enginu vrátí strukturu s ukazateli na konkrétní implementaci. Funkce pak uživatel volá přes tuto strukturu vždy stejně, přestože implementace se může pro různé tabulky lišit.

Každý engine musí implementovat následující operace:

1. `insert_row`
2. `modify_row`
3. `delete_row`
4. `set_default`
5. `clear_default`
6. `insert_all`
7. `reset`
8. `enable`
9. `disable`

Pokud engine nebude některou z operací podporovat, bude mít k dispozici předpřipravenou funkci stejné hlavičky a jména s prefixem `unsupported_`, která nic neprovede.

8. Testování

Každý software by měl být nějakým způsobem testován, obvzlášť pokud má být přikompilován do kernelu. O testovacím frameworku zatím nebylo rozhodnuto. K použití se nabízí testovací C++ knihovna od Googlu. Nevýhodou však je, že by to zbytečně zanášelo C++ kód do čistého kódu v C. Proto jsou testy zatím tvořeny

pomocí maker v C. Test je v podstatě program v C se staticky přilinkovaným ovladačem libp4dev.

9. Závěr

Cílem práce bylo seznámit se staršími verzemi ovladače libp4dev a provést novou implementaci podle nového návrhu. Implementačně jsem stihl device tree parser modul s testy a základ pro entitu tabulek s enginy. Jelikož byl projekt P4 komercializován a předán firmě Intel, práce už nebude dál pokračovat.

Poděkování

Rád bych poděkoval Pavlu Benáčkovi, který byl skvělý vedoucí, vždy ochotný pomoci. Mé díky patří také Tomáši Martínkovi, který mi poskytl zpětnou vazbu k technické zprávě.

Literatura

[1] CESNET. *NFV200 – Platforma pro akceleraci virtualizace funkcí sítě*. Duben 2020. Dostupné z: https://www.cesnet.cz/projekty/nfv200/.

[2] CONSORTIUM, T. P. L. *P416 Language Specification*. Květen 2017. Dostupné z: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html.

[3] JAWOROWSKI, R. *BSD Kernel Interfaces Manual*. Březen 2019. Dostupné z: https://www.freebsd.org/cgi/man.cgi?query=FDT.