VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

OVLADAČ NETDEV PRO AKCELERAČNÍ KARTY COMBO

NETDEV DRIVER FOR ACCELERATION COMBO CARDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DOMINIK TRAN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAN KUČERA**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Tran Dominik**
Obor: Informační technologie
Téma: **Ovladač netdev pro akcelerační karty COMBO**
Netdev Driver for Acceleration COMBO Cards
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s FPGA síťovými kartami rodiny COMBO a s aktuálním programovým rozhraním SZE2 pro příjem ethernetových rámců ze sítě.
2. Navrhněte ovladač těchto síťových karet umožňující pro příjem i odesílání dat nezávisle využívat také standardní síťové rozhraní jádra systému Linux.
3. Proveďte implementaci ovladače podle návrhu.
4. Ověřte funkčnost síťového rozhraní na službách přímo poskytovaných jádrem systému (např. ICMP, ARP, NDP, DHCP). Zjistěte výkonové parametry implementovaného řešení.
5. V závěru diskutujte dosažené výsledky a možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kučera Jan, Ing.**, UPSY FIT VUT
Konzultant: Špinler Martin, Ing., CESNET
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Tato práce se zabývá návrhem a implementací ovladače nad FPGA síťovými kartami rodiny COMBO, který by umožnil příjem a odesílání paketů prostřednictvím standardního síťového rozhraní jádra Linux. Sdružení CESNET vyvíjí zařízení typu Protector pro ochranu proti amplifikačním (D)DoS útokům využívající akcelerační karty COMBO pro dosažení vysokého výkonu. Přenos síťových dat mezi kartou a řídicí aplikací je realizován rozhraním SZE2, které umožňuje rychlé zpracování dat mj. díky obcházení síťové vrstvy jádra. Zařízení typu Protector však musí podporovat standardní síťové protokoly, jejichž vlastní implementace přímo nad rozhraním SZE2 by byla velmi náročná. Místo toho se nabízí využití síťové vrstvy v linuxovém jádře, která se pro dosažení vysoké výkonnosti běžně obchází. Vytvořením ovladače síťového zařízení lze využít služeb síťové vrstvy jádra včetně standardních síťových aplikací. Na základě nastudování zejména rozhraní SZE2 a principů vývoje ovladačů byl navržen a následně i úspěšně implementován ovladač síťového zařízení, který byl otestován z pohledu funkčnosti i výkonnosti. Nad rámec zadání byl implementován stejný ovladač nad novějším rozhraním NDP a aplikace pro řízení akcelerované přeposílání paketů.

Abstract

This thesis deals with the development of the network device driver for the FPGA network COMBO cards, which should enable receiving and sending packets through standard network interface of Linux kernel. CESNET is developing a device called DDoS Protector for protection against an amplification (D)DoS attacks, which uses COMBO cards to achieve high performance. A SZE2 interface is used for high speed transfers of network data between COMBO card and a controlling software application, using technique of bypassing kernel network stack and other methods. DDoS Protector has to support standard network protocols, whose implementation directly on top of the SZE2 is very difficult. Instead, using kernel network stack, which is, by default, bypassed to achieve high performance, is much easier to implement and supports all sorts of protocols. Creation of the network device driver enables us to use kernel network stack and other network applications for COMBO cards. Based on the study of SZE2 interface and driver development, I designed and then successfully implemented network device driver. Driver was tested to ensure standard protocols work. It was also tested from the performance point of view. I have also developed the same type of driver for the newer interface – NDP and an application for an accelerated packet forwarding, both of which are functional and were not part of the thesis specification.

Klíčová slova

CESNET, COMBO, SZE2, NDP, DDoS Protector, ovladač, síťové rozhraní, Linux, směrování, BGP

Keywords

CESNET, COMBO, SZE2, NDP, DDoS Protector, driver, network interface, Linux, routing, BGP

Citace

TRAN, Dominik. *Ovladač netdev pro akcelerační karty COMBO*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kučera

Ovladač netdev pro akcelerační karty COMBO

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Kučery. Další informace mi poskytl Ing. Martin Špinler. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Dominik Tran
16. května 2018

Poděkování

Velmi rád bych poděkoval vedoucímu práce Ing. Janu Kučerovi za odborné vedení, ochotu a trpělivost. Také bych rád poděkoval konzultantovi Ing. Martinu Špinlerovi za pomoc a cenné rady.

Obsah

Kapitola 1

Úvod

Počítače, čímž se kromě laptopů myslí i chytré mobilní telefony, jsou dnes nedílnou součástí každodenního života. Jejich důležitou vlastností je schopnost komunikovat s ostatními zařízeními přes síť Internet. Počet zařízení připojených k Internetu přitom narůstá – nejenom zvětšujícím se počtem populace, která má přístup k Internetu, ale i vlivem tzv. Internetu věcí, známého také pod zkratkou IoT (Internet of Things), kdy jsou do Internetu připojena různá spotřební zařízení, jako například lednička, kamery či různé senzory. Zvětšuje se také konzumace multimediálních dat. Tyto jevy vedou k rostoucímu objemu dat, která se přes síť každou sekundu přenáší. Pro zajištění určité kvality služeb je potřeba stále výkonnějších síťových zařízení, zejména směrovačů.

Kromě nárůstu objemu dat a rychlostí v sítích je však potřeba se zabývat také bezpečností sítí a chránit je tak před neustále hrozícím nebezpečím. Útoky typu DoS, respektive jejich distribuovaná a také častější varianta DDoS (Distributed Denial of Service) jsou jednou z častých hrozeb. Cílem takových útoků je poškození nebo vyřazení určitého systému z činnosti a jeho znepřístupnění ostatním legitimním uživatelům. Útoky typu DoS mohou nabírat různých podob, například cílem SYN flood útoků je vyčerpání zdrojů cílového serveru, zatímco amplifikační volumetrické útoky mohou cílit na zahlcení kapacity síťové linky připojené koncové sítě. Takové typy útoků je přitom třeba řešit už na úrovni nadřazené sítě. Dnešní počítačové sítě přitom pracují na velmi vysokých rychlostech – desítky až stovky Gb/s.

Sdružení CESNET aktuálně vyvíjí zařízení DDoS Protector pro ochranu přesně proti takovým útokům. Pro dosažení vysokého výkonu je zařízení postaveno nad FPGA akceleračními kartami COMBO, kdy se úlohy rozdělí na softwarovou (řídicí) část a hardwarovou (výkonnou) část realizovanou kartami COMBO. Pro příjem paketů ze síťové karty s technologií FPGA a jejich další zpracování v softwaru se využívá rozhraní SZE2, které umožňuje zpracování velkého objemu dat, a to díky mechanismu DMA, obcházením síťové vrstvy jádra Linux a zero copy mechanismu. Avšak i zařízení typu Protector se musí v síti chovat jako standardní síťové zařízení, tzn. podporovat standardní protokoly ARP, DHCP, BGP, ICMP. Vlastní implementace takové podpory přímo nad rozhraním SZE2 by ale byla velmi náročná. Místo toho se nabízí využití existující implementace v linuxovém jádře, která je jeho standardní součástí a která se pro dosažení vysoké výkonnosti při příjmu síťových dat běžně obchází. V případě servisních protokolů ARP, DHCP, BGP či ICMP a dalších se však jedná o nekritické úlohy, tudíž není potřebné řešit vysoký výkon příjmu paketů, jako v případě blokování útoku.

Cílem této bakalářské práce je tak vytvořit ovladač pro standardní síťové rozhraní nad připojenou kartou COMBO, který se využije pro příjem paketů pro nekritickou část Protec-

toru. Zde se využije již naimplementované podpory protokolů v jádře kernel ke zpracování paketů. Nad takovým síťovým rozhraním bude možné použít například standardní nástroj `ping`. Paralelně s takovým rozhraním je však třeba zachovat možnost zpracovávat pakety také pomocí rychlého rozhraní SZE2, pro potřeby čištění provozu.

V rámci práce byl proveden návrh a implementace ovladače pro vysokorychlostní rozhraní SZE2. Nad rámec práce byl poté navržen a implementován tentýž ovladač také pro novější generaci tohoto rozhraní (NDP), které bylo představeno až v průběhu řešení bakalářské práce. Dále byl rovněž nad rámec zadání implementován démon, který umožňuje čtení směrovacích informací prostřednictvím protokolu BGP a jejich synchronizaci s firmwarovou jednotkou na akcelerační síťové kartě COMBO. Vytvořený ovladač i aplikace jsou plně funkční. Korektní funkce síťového rozhraní byla podle zadání ověřena pro protokoly ICMP, ARP, NDP, DHCP a BGP a dále byla experimentálně změřena také dosažená propustnost vytvořeného řešení.

Práce je logicky rozdělena do několika kapitol. Kapitola 2 se věnuje teoretickému základu, který pokrývá vrstvový model sítě, zařízení DDoS Protector, vývoj ovladačů v linuxovém jádře a akcelerační karty COMBO včetně popisu rozhraní SZE2. Navazující kapitola 3 v úvodu popisuje jedno z možných zapojení DDoS Protectoru, na kterém vysvětluje motivaci pro vznik ovladače. Dále se kapitola zabývá možnými přístupy pro příjem paketů nad rozhraním SZE2, návrhem ovladače i návrhem démona. Kapitola 4 popisuje implementaci ovladače nad rozhraním SZE2, novějším NDP a implementaci démona. Další kapitola 5 se věnuje testování a vyhodnocení obou ovladačů, a to z funkčního i výkonového hlediska včetně jejich porovnání. Poslední kapitola 6 shrnuje dosažené výsledky práce.

Kapitola 2

Teoretický rozbor

V této kapitole budou podrobněji popsány části a pojmy, se kterými se bude pracovat v navazujících kapitolách. Jejich porozumění je nutnou podmínkou pro hladkou orientaci v následujících kapitolách této práce. V podkapitole 2.1 se čtenář seznámí se stručným principem fungování komunikace po síti a standardními protokoly. Jedná se o nutný teoretický základ. V podkapitole 2.2 jsou stručně rozebrány akcelerační karty COMBO – jejich struktura a použité technologie. V podkapitole 2.3 je detailněji popsána činnost zařízení DDoS Protector. V podkapitole 2.4 se čtenář dozví základní informace o jádře Linux a problematice vývoje ovladačů se zaměřením na ovladače pro síťová zařízení. V poslední podkapitole 2.5 je popsáno, jakým způsobem funguje rozhraní SZE2 z pohledu hardwarové i softwarové části.

2.1 Principy síťových zařízení, standardní protokoly

Následující část věnovaná principům počítačových sítí čerpá informace primárně z 1. kapitoly knihy Síťové aplikace a jejich architektura [14] a další literatury [5, 23].

Architektura počítačových sítí je dnes popisována z pohledu tzv. vrstvového modelu sítě. Vrstvový model počítačové sítě se skládá z několika vrstev, kde každá vrstva plní nějaký účel (poskytuje určitou funkcionalitu) a vrstvy na sebe přímo navazují. Služby jednotlivých vrstev jsou implementovány příslušnými protokoly. Vyšší vrstvy využívají služeb nižších vrstev, které jsou pro ně transparentní. Dnes jsou používané dva vrstvové modely sítě – referenční model ISO/OSI a internetový model TCP/IP.

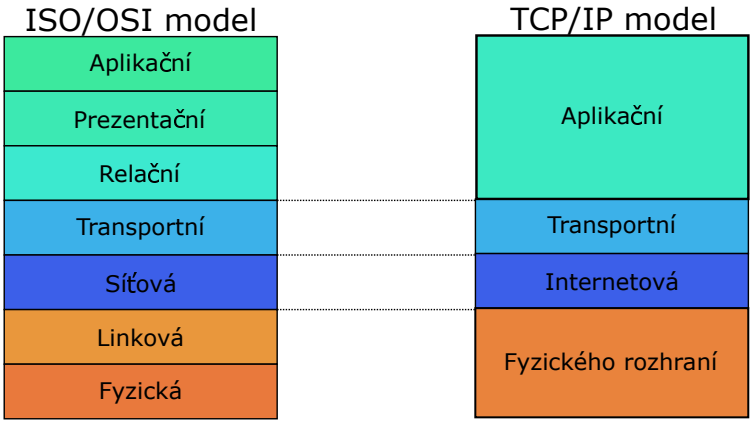
Model OSI (Open Systems Interconnection) vytvořený Mezinárodní organizací pro normalizaci (ISO) obsahuje 7 vrstev a je často používán ve výuce jako názorný příklad komunikace v počítačových sítích. Struktura vrstev modelu OSI je následující (od shora dolů):

- *Aplikační vrstva* – nejvyšší vrstva, komunikuje s uživatelskými procesy a umožňuje jim využít služeb nižších vrstev.
- *Prezentační vrstva* – transformuje data do požadovaného formátu aplikace. Zahrnuje kódování, kompresi a odlišné formáty dat jako například ASCII (American Standard Code for Information Interchange) nebo binární data.
- *Relační vrstva* – slouží k vytváření, udržování a ukončování sezení (relací) mezi komunikujícími aplikacemi.
- *Transportní vrstva* – zajišťuje spolehlivý přenos dat mezi koncovými stanicemi. Rozděluje velká data na několik menších částí (segmentace).

- *Síťová vrstva* – zajišťuje adresování a směrování dat.
- *Linková vrstva* – řídí přenos dat mezi dvěma lokálními datovými uzly, zajišťuje adresaci, kontroluje a opravuje chyby při přenosu.
- *Fyzická vrstva* – nejnižší vrstva, zajišťuje odeslání a příjem dat po fyzickém médiu. Definuje fyzické vlastnosti linky, např. počet pinů konektoru nebo velikost napětí.

Celý model OSI se však v praxi moc nerozšířil. Využila se pouze jeho část.

Model TCP/IP je dnes reálně používaný model pro komunikaci v Internetu. Oproti modelu ISO/OSI je jednodušší, protože definuje pouze 4 vrstvy. Oba modely jsou si podobné – lze říci, že model TCP/IP je zjednodušená varianta modelu OSI. Srovnání obou modelů je na obrázku 2.1. Model obsahuje 4 vrstvy: aplikační, transportní, internetovou a



Obrázek 2.1: Srovnání modelu OSI a modelu TCP/IP.

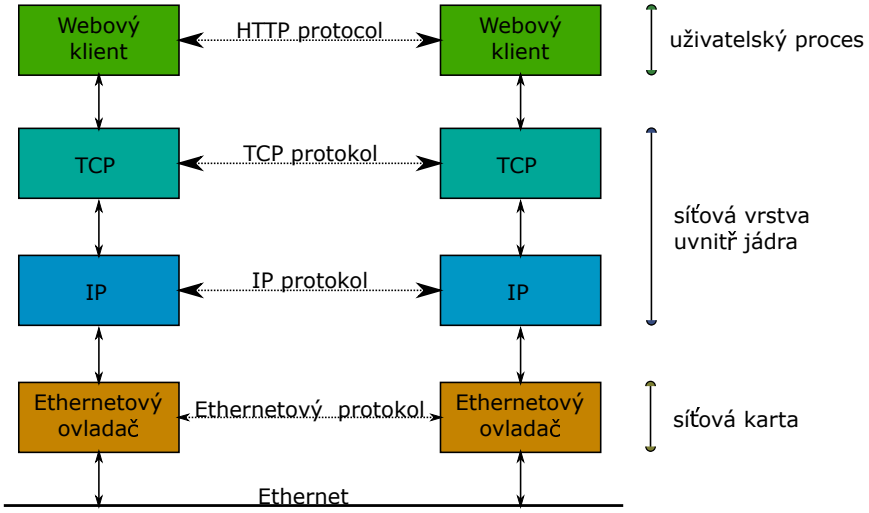
vrstvu síťového rozhraní. Obrázek 2.1 ukazuje logické sloučení některých vrstev OSI modelu do vrstev TCP/IP modelu.

Při průchodu dat od nejvyšší vrstvy k nejnižší (odesílání) dochází k zapouzdření. Jedná se proces, kdy si jednotlivé vrstvy k samotným datům přidávají svá vlastní metadata ve formě hlaviček. Naopak při příjmu dat dochází k rozbalování dat, kdy jednotlivé vrstvy zpracují data ze své hlavičky a tu nakonec odstraní. K cílovému procesu tak nakonec přijdou pouze samotná data. Následuje přehled činnosti jednotlivých vrstev:

- *Aplikační vrstva* – adresace záleží na aplikaci. Vznikají data.
- *Transportní vrstva* – vytváří logické spojení mezi procesy, které identifikuje číslem portu. Vzniká TCP/UDP[1] paket, také označovaný jako TCP segment nebo UDP datagram.
- *Internetová vrstva* – vytváří logické spojení mezi uzly, které identifikuje IP adresou. Vzniká IP datagram, také označovaný jako paket.
- *Vrstva síťového rozhraní* – adresuje zařízení v lokální sítí pomocí MAC adresy. Vzniká rámec.

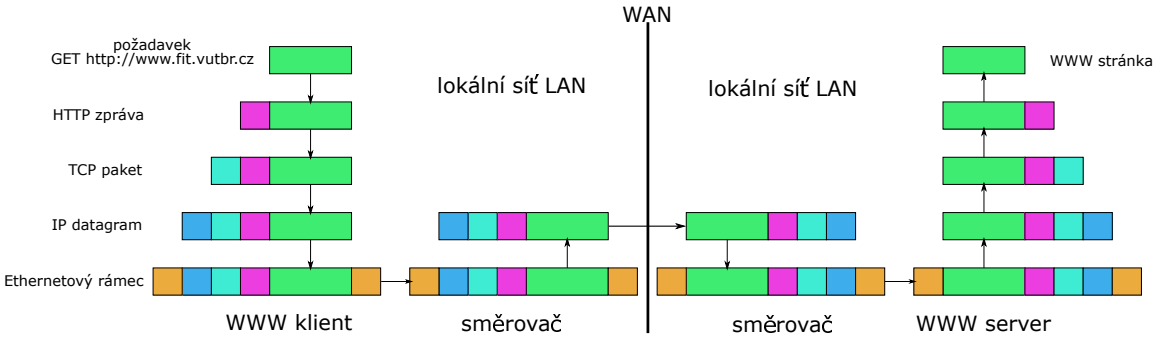
Na obrázku 2.2 je znázorněna webová komunikace mezi jednotlivými vrstvami. Zároveň na druhém obrázku 2.3 je znázorněno zapouzdřování dat při odesílání a příjmu při téže komunikaci.

[1] User Datagram Protocol



Obrázek 2.2: Webová komunikace na jednotlivých vrstvách TCP/IP modelu.

Jedná se komunikaci webové služby využívající protokol HTTP (Hypertext Transfer Protocol). Klient chce odeslat požadavek na získání obsahu webové stránky www.fit.vutbr.cz. Požadavek je z webového klienta uložen do formátu aplikačního protokolu HTTP. Poté je předán transportní vrstvě. Ta k datům přidá svoji hlavičku obsahující čísla portů a vzniká TCP (Transmission Control Protocol) paket, který následně dostane internetová vrstva. Na ní je přidána hlavička obsahující IP adresy a vzniká IP (Internet Protocol) datagram. IP datagram je předán fyzické vrstvě, která přidá svoji hlavičku obsahující MAC (Media Access Control) adresy a vzniká rámec. Ten je nakonec vyslán na fyzické médium ve formě jednotlivých bitů. Směrovače na cestě k cíli využívají informaci o cílové IP adrese, nutně tedy musí provádět rozbalení a znovu-zapouzdření. Když cílové zařízení obdrží rámec, zkontroluje, jestli cílová MAC adresa odpovídá jeho MAC adrese. Pokud ano, hlavičku odstraní a vzniklý IP datagram předá internetové vrstvě. Na této vrstvě je zkontrolována cílová IP adresa, a pokud se shoduje, odstraní se hlavička a vzniklý TCP paket je předán transportní vrstvě. Transportní vrstva zkontroluje, zdali je proces s cílovým číslem portu spuštěn. Pokud ano, odstraní hlavičku a data předá procesu (službě, aplikaci).



Obrázek 2.3: Zapouzdření na jednotlivých vrstvách TCP/IP modelu.

Bylo zmíněno, že vlastnosti vrstev jsou implementovány různými protokoly. U zařízení DDoS Protector, pro které se v této práci vytváří implementace ovladače pro síťová zařízení, vznikl požadavek na schopnost se v síti chovat jako standardní síťové zařízení, tzn. podporu

různých standardních protokolů. Konkrétně se jedná se o protokoly DHCP, ICMP, ARP, NDP a BGP, které nyní budou stručně popsány:

- *DHCP* (Dynamic Host Configuration Protocol [4]) je aplikační protokol používaný pro automatickou konfiguraci zejména IP adresy, masky sítě, DNS serveru a výchozí brány. Tyto údaje jsou nutné pro připojení do sítě. Využití služeb DHCP serveru je tak jednou z prvních věcí, kterou zařízení využije. DHCP využívá porty 67 (server) a 68 (klient) [4]. Přidělené údaje jsou časově omezené a je nutné je periodicky od DHCP serveru obnovovat.

- *ARP* (Address Resolution Protocol [18]) je komunikační protokol pracující na internetové a fyzické vrstvě ve verzi IPv4. Jeho cílem je zajistit mapování IP adresy na MAC adresu. Například po čerstvém získání IP adresy výchozí brány od DHCP protokolu potřebuje zařízení ještě zjistit její MAC adresu, aby s ní mohlo komunikovat v lokální sítí. Zařízení v takovém případě vyšle ARP dotaz na všesměrovou MAC adresu a uvědomí tak všechny zařízení v lokální síti, včetně výchozí brány. Ta odešle ARP odpověď a zařízení si odpověď uloží do své ARP cache [21].

- *NDP* (Neighbor Discovery Protocol [15]) je komunikační protokol pracující na internetové vrstvě a vrstvě síťového rozhraní. Jedná se náhradu ARP protokolu pracující ve verzi IPv6. Oproti ARP obsahuje více funkcí – například obsahuje zjišťování nedostupnosti sousedních uzlů NUD (Neighbor Unreachability Detection), čímž se zlepší odolnost doručení paketu při selhání směrovačů [15]. Protokol NDP je součástí ICMPv6.

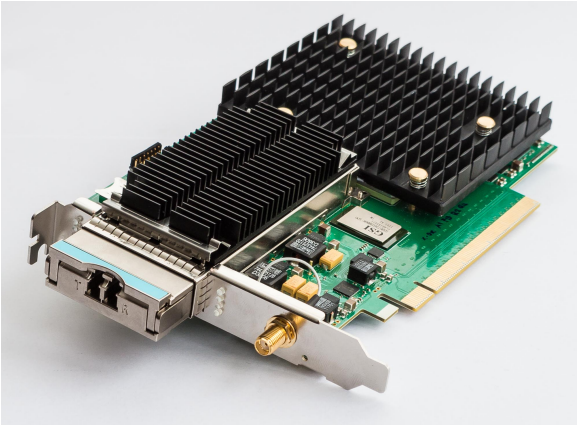
- *ICMP* (Internet Control Message Protocol [19]) je protokol pracující na internetové vrstvě. Je používán k posílání chybových nebo informačních zpráv. Jeho obsah je zapouzdřen přímo v IP datagramu [19]. Známý nástroj `ping` využívá ICMP zpráv Echo Request (požadavek) a Echo Reply (odpověď) pro komunikaci s cílovým zařízením.

- *BGP* (Border Gateway Protocol [20]) je dynamický směrovací protokol, který se používá k výměně směrovacích informací mezi různými autonomními systémy (např. poskytovali internetu). Dnes je používán pro směrování mezi páteřními sítěmi. Hraniční směrovače šíří přijaté informace (z jiných autonomních systému) pomocí iBGP (Internal BGP) varianty uvnitř svého autonomního systému [1]. Opakem je eBGP (External BGP) pro komunikaci mezi různými AS.

Výše zmíněné protokoly kromě BGP mají v linuxových systémech standardní implementaci – ať už uvnitř samotného jádra, nebo ve formě nějakého systémového démona (DHCP). BGP ani další směrovací protokoly implementovány nejsou, protože standardní síťové zařízení jako například osobní počítač takové protokoly ke své funkci nepotřebuje. Zařízení DDoS Protector ale musí být schopné pasivně přijímat směrovací informace, aby mohlo bez manuálního zásahu posílat vyčištěný provoz zpět do cílové sítě (detailněji se Protectoru věnuje podkapitola 2.3). Podporu BGP nicméně lze najít v podobě dostupných softwarových balíčků. Jedním z nich je BIRD (BIRD Internet Routing Daemon), nyní vyvíjený Laboratořemi CZ.NIC [3]. Jedná se o směrovacího démona pro dynamické směrování IP protokolu vyvíjeného hlavně pro linuxové a BSD operační systémy.

2.2 Akcelerační karty COMBO

Akcelerační karty rodiny COMBO jsou určeny pro zpracování síťových dat. Srdcem těchto karet je FPGA čip, obsahující firmware. Karta dále obsahuje paměti (statické i dynamické), zdroj hodin, PCI-Express sběrnici a konektory, napájení a další součásti [6]. Na obrázku 2.4 je zachycen fyzický vzhled 100 gigabitové karty.



Obrázek 2.4: Karta COMBO-100G [13].

Následující části věnující se technologii FPGA, standardu PCI-Express a DMA přenosům vychází primárně z [6]. Na podpůrné zdroje jsou v textu uvedeny odkazy.

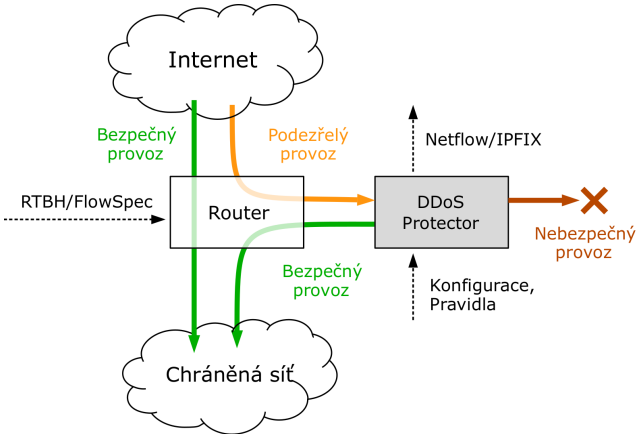
FPGA je technologie programovatelných hradlových polí. Jedná se o integrovaný logický obvod složený z elementárních bloků, které je možné mezi sebou libovolně propojit a tím lze docílit vytvoření téměř jakéhokoliv číslicového obvodu [25]. Pro popis takových struktur se využívají speciální programovací jazyky. Známé a často používané jsou jazyky Verilog (převažuje v USA) a VHDL (VHSIC Hardware Description Language). V rámci projektu Liberouter se konkrétně používá VHDL. Vytvořený program se vysyntetizuje do cílové technologie a vznikne tzv. bitstream. Ten se nahraje do FPGA čipu a nazývá se firmware. Současné karty COMBO využívají FPGA čipy Virtex7 a Virtex UltraScale+ [12] od firmy Xilinx [26].

S hostitelským systémem komunikují karty COMBO skrze sběrnici PCI-Express. PCI-Express je přesněji standard sériové systémové sběrnice z roku 2004. Umožňuje plně duplexní přímý přenos mezi dvěma koncovými body. Body jsou propojeny tzv. linkou, sestávající se ze dvou vodičů [24]. Linek může být více, standard definuje 1, 2, 4, 8 a 16 linek. Propustnost u třetí generace (PCI-Express 3.0) na jedné lince dosahuje až 8 Gb/s. Při využití 16 linek lze dosáhnout propustnosti až 128 Gb/s. Pro akcelerační kartu, která musí být schopná zpracovávat síťový provoz o rychlosti až 100 Gb/s, je taková propustnost dostačující.

U karet COMBO se pro přenos dat po sběrnici PCI-Express mezi pamětí FPGA čipu a pamětí RAM hostitelského systému využívá princip DMA (Direct Memory Access). DMA je technika pro přímý přístup do paměti RAM. Umožňuje vstupně/výstupním zařízením přístup do paměti RAM bez nutnosti aktivní účasti procesoru. Bez použití mechanismu DMA je procesor typicky zcela zaneprázdněn po čas čtení/zápisu do paměti. Uvolnění procesoru od takové činnosti se projeví výrazným (zvlášť při častém čtení/zápisu) nárůstem výpočetního výkonu, jelikož se procesor může věnovat jiným operacím.

2.3 DDoS Protector

DDoS Protector je aktivní síťové zařízení vyvíjené sdružením CESNET v rámci projektu Liberouter [11]. Jeho úkolem filtrovat různé typy DDoS útoků. Cílem (D)DoS útoků je obecně vyčerpat prostředky oběti, která se pak legitimním uživatelům a službám jeví jako nedostupná. DDoS Protector je zařízení zaměřující se primárně na volumetrické útoky. Ty se snaží o vyčerpání konektivity oběti, tzn. o zahlcení jejího přípojného spoje dostatečně velkým objemem dat za jednotku času. Oběť se proti útokům takového typu nemůže sama bránit a proto je potřeba útok řešit už v síti poskytovatele. Protector může být tak zapojen přímo do infrastruktury sítě poskytovatele, aby pomohl efektivně ochránit připojenou koncovou síť. Jedno z možných zapojení je znázorněno na obrázku 2.5. Protector je zapojen v páru se směrovačem, který do něj bude směrovat podezřelý provoz. Protector by měl nebezpečný provoz (například probíhající amplifikační DNS útok) utlumit, zatímco legitimní provoz by se měl beze změny vrátit zpátky do sítě přes přidružený směrovač. Zpoždění legitimních paketů je minimální, řádově v rámci mikrosekund [27].



Obrázek 2.5: Doporučené zapojení DDoS Protectoru v síti [27].

Způsob filtrace provozu na síti závisí na sadě pravidel definované správcem. Každé pravidlo musí obsahovat [11]:

- Prefix (rozsah IP adres), který má být chráněn,
- podmínky, resp. upřesnění nebezpečného provozu (zdrojové a cílové porty, protokol, ...),
- práh (hranici) v podobě maximálního počtu paketů a/nebo bitů za sekundu, který by se neměl překročit, a
- limit, na který by se měl provoz do chráněné sítě snížit v případě překročení prahu.

Kontrole je podroben každý paket. Pokud informace z hlaviček paketu odpovídají některému pravidlu, je zdrojová IP adresa (pokud již není zaznamenána) uložena do vnitřních struktur Protectoru a aktualizuje se počet přenesených paketů a bitů. Tyto statistiky jsou u všech pravidel periodicky kontrolovány. V případě překročení nastaveného prahu se začne zahazovat (blokovat) provoz směřující do chráněné sítě a to takovým způsobem, aby se provoz snížil alespoň na stanovený limit. Přesněji se postupně začne blokovat provoz z těch zdrojových IP adres, které přenesly největší objem dat do chráněné sítě [11]. Filtrace

provozu je poměrně jemná, protože většina legitimního provozu, který standardně nebývá z hlediska objemu přenesených dat moc velký, nebude zablokována.

Následuje příklad jednoduchého pravidla, které chrání prefix 87.1.0.0/16 od UDP provozu se zdrojovým portem 53, což jsou příznaky DNS amplifikačního útoku. Práh je nastaven na 25 Gb/s a limit na 12 Gb/s:

```
dst net 87.1.0.0/16 protocol UDP src port 53 threshold 25Gbps limit 12Gbps
```

Architektura Protectoru je rozdělena na hardwarovou (výkonnou) a softwarovou (řídící) část [10]. Softwarová část komunikuje s uživatelem, načítá a spravuje pravidla, provádí kontrolu prahu, komunikuje s hardwarem (přes speciální uživatelské knihovny) a vykonává mnoho dalších funkcí. Hardwarová část je kromě standardního serverového hardwaru (procesor, paměti RAM, pevné disky...) tvořena akceleračními kartami COMBO. Karty a jejich součásti byly popsány v podkapitole 2.2. Obsahují filtry pro blokaci nebezpečných IP adres, spravují čítače a provádí příjem a odeslání paketů [10]. To zahrnuje mj. kontrolu TTL (Time to Live) nebo modifikaci MAC adres. Na obrázku 2.6 je ukázka celého zařízení DDoS Protector na platformě SuperServer firmy Supermicro.

Obrázek 2.6: DDoS Protector na 1U Supermicro SuperServer 1028U-TRT+ platformě [11].

Maximální propustnost Protectoru je 100 Gbit/s, resp. 150 Mp/s (150 miliónů paketů za sekundu) [11]. Takto vysoké propustnosti je dosaženo využitím hardwarové akcelerace karet COMBO nad rozhraním SZE2. Protector podporuje IPv4 i IPv6, je schopen pracovat až s 3000 pravidly naráz a v případě útoku blokovat až 16 000 IP adres (v budoucnosti se plánuje až 100 tisíc [27]). Protector se zaměřuje primárně na volumetrické útoky, ale je navržen a postaven modulárně. Do budoucna je tak možné vytvářet funkcionalitu i proti dalším typům útoků. Kromě výše uvedeného amplifikačního příkladu v Protectoru existuje i modul pro mitigaci TCP SYN flood útoků. Ty necílí na zahlcení koncové linky, ale přímo na vyčerpání dostupných zdrojů oběti.

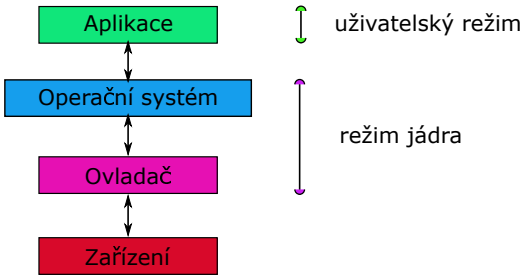
2.4 Linux Kernel, vývoj ovladačů

Tato podkapitola čerpá informace z knihy Jádro systému Linux [9]. Linux je monolitické jádro[2] operačního systému, šířené jako svobodný software (pod licencí GNU GPLv2). Autorem první veřejné verze 0.01 je Linus Torvalds, který ji vydal v roce 1991. Linux se neustále vyvíjí a využívá moderní technologie. Je flexibilní – využívá se pro běh serverů, superpočítačů, vestavěných zařízení[3] i na klasických desktopech. Svobodná licence umožňuje komukoliv studovat a libovolně upravovat zdrojové kódy, které jsou napsané v jazyce C.

Ovladač slouží jako most mezi zařízením, které chceme ovládat a operačním systémem, jehož služby využívá uživatelská aplikace přes standardní rozhraní. V Linuxu se jednotlivé ovladače mohou ve formě modulů jádra dynamicky zavádět a odstraňovat. Na obrázku 2.7 je znázorněn vztah mezi uživatelskou aplikací, operačním systémem, ovladačem zařízení

[2] jádro samotné je mrtvý kód, procesy volají jeho funkce
[3] embedded systems

a samotným zařízením. Ovladač (resp. jeho kód) pracuje v režimu jádra, který se liší od uživatelského režimu, v němž běží všechny uživatelské procesy.



Obrázek 2.7: Princip komunikace mezi hardwarem a aplikacemi.

V uživatelském režimu platí různá omezení (omezená instrukční sada, operace s pamětí, ...). Proces nemá právo přistoupit za hranice paměti, která mu byla operačním systémem přidělena. V režimu jádra taková omezení neplatí. Programátor jádra/ovladače má tak v některých ohledech větší svobodu, na druhou stranu to přináší větší zodpovědnost a disciplínu. Zároveň musí počítat s dalšími aspekty, které odlišují vývoj ovladače od uživatelského programu, jako například:

- Omezené prostředky – paměť, čas procesoru. Pomalý kód v jádře by mohl výrazně omezit výkon celého systému.
- Omezené rozhraní – procesy mohou s jádrem komunikovat přes systémová volání nebo jiné prostředky (např. `ioctl`).
- Celé jádro sdílí jediný adresní prostor – některá část jádra by mohla přepsat data jiné části, což by mohlo vést až k pádu systému.
- Omezená množina funkcí – nejsou k dispozici rozsáhlé uživatelské knihovny.

Každý modul jádra má nějaký účel. Ovladače zařízení lze rozdělit do 3 základních kategorií:

- Ovladače znakového zařízení – poskytují přístup proudovým způsobem.
- Ovladače blokových zařízení – poskytují přístup přes adresovatelné bloky.
- Ovladače síťových zařízení – poskytují přístup přes síťovou vrstvu.

Znaková zařízení jsou díky sériovému přístupu dat vhodná například pro (virtuální) terminály, klávesnice nebo zvuková (audio) zařízení. Bloková zařízení často odpovídají úložným zařízením jako je pevný disk, CD/DVD nebo flash paměti. Tato práce se bude podrobněji zabývat ovladači síťových zařízení. Významnou vlastností síťových zařízení je, že na rozdíl od blokových a znakových zařízení vede mezi uživatelským procesem a zařízením poměrně dlouhá cesta. K samotným datům se při odesílání na síť postupně přidávají různé hlavičky, a naopak při příjmu dat se hlavičky kontrolují a následně odstraňují – to zajišťují ovladače pro jednotlivé síťové vrstvy a protokoly. V rámci této práce by měl vzniknout ovladač síťového zařízení nad kartami COMBO tak, aby bylo možné využít služeb právě zmíněných síťových vrstev a protokolů implementovaných v jádře.

Ovladač síťového zařízení má 2 základní úkoly: odesílání a příjem. Pokud uživatelský proces zavolá funkci pro odeslání dat, data prochází síťovou vrstvou až do ovladače. Ten

přidá hardwarovou hlavičku a paket odešle. Vlastní odeslání většinou spočívá v nastavení a spuštění DMA. Síťové zařízení k datům přistoupí a odešle je přes fyzické médium. Příjem dat je asynchronní záležitost – zařízení příchod nových dat signalizuje přerušením. Ovladač připraví a spustí DMA přenos ze zařízení do paměti systému. Pak předá přijatý paket síťové vrstvě ke zpracování. Příjem je náročnější než odesílání, protože ne vždy musí být dostatek místa pro nové pakety. U vysokorychlostních síťových zařízení by generování přerušení při příchodu každého paketu výrazně omezilo výkon systému. Proto se používá odlišný způsob příjmu dat – dotazový režim (polled mode). Při dotazovém režimu se funkce *poll()* provádí periodicky nebo ve vhodných momentech (například když jádro nevykonává žádnou důležitou operaci) a pokud přišla nová data, tak se zpracují.

Samotné síťové zařízení je v jádru reprezentováno strukturou `struct net_device`. Tato struktura obsahuje asi 35 položek. Některé z nich jsou jméno (např. `eth0`), číslo přerušení nebo MAC adresa. Jádro dále využívá strukturu `struct sk_buff`, ve které se uchovávají samotná data paketu a s níž pracuje síťová vrstva jádra. Následuje ukázka kódu [9] pro velice jednoduchý příjem paketu:

```
static int mynet_rx(struct net_device *dev, void *data, size_t len) {
   struct sk_buff * skb = dev_alloc_skb(len);

   skb->dev = dev;
   skb->protocol = eth_type_trans(skb,dev);   // Ethernet
   skb->ip_summed = CHECKSUM_UNNECESSARY;
   memcpy(skb_put(skb, len), data, len);
   netif_rx(skb);

   return 0;
}
```

Funkce alokuje buffer pro paket a vyplní některé jeho datové položky. Pak zkopíruje data příchozího paketu do bufferu a ten předá síťové vrstvě jádra.

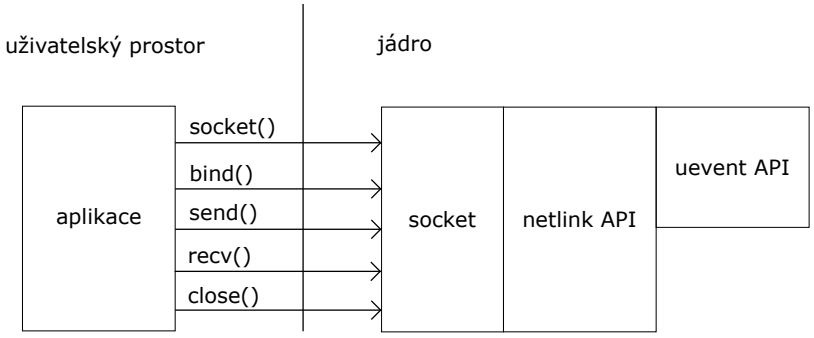
Ukážeme si ještě jednoduchý kód [9] pro inicializaci a odebrání ovladače pro síťové zařízení.

```
static struct net_device * mynetdev = NULL;

static int mynet_open(struct net_device *dev) {...}
static int mynet_stop(struct net_device *dev) {...}
static int mynet_tx(struct sk_buff *skb, struct net_device *dev ) {...}

static void mynet_setup(struct net_device *dev) {
   dev->open = mynet_open;
   dev->stop = mynet_stop;
   dev->hard_start_xmit = mynet_tx;
}
static int __init mynet_init(void) {
   mynetdev = alloc_netdev(0, "mynet", mynet_setup);
   register_netdev(mynetdev);

   return 0;
```

```
}

static void __exit mynet_cleanup(void) {
   unregister_netdev(mynetdev);
   free_netdev(mynetdev);
}
```

Inicializační funkce `mynet_init` se zavolá při načtení ovladače jako modulu (příkaz `insmod`). Funkce alokuje prostředky pro síťové zařízení `net_device`. Třetí parametr je ukazatel na vlastní funkci, která síťovému zařízení nastaví ukazatele na funkce pro standardní operace jako otevření (`ip link set dev mynet up`), zavření nebo vysílání paketů. Poté zařízení zaregistruje. Od té chvíle je možné zařízení spatřit jako standardní síťové rozhraní pod jménem *mynet*, například při výpisu nástroje ip – `ip link show`, spolu s ostatními síťovými rozhraními, jako je `eth0`. Předvedené části kódu ukazují, že vytvoření jednoduchého ovladače se tak zásadně neliší od vytvoření klasického programu. Vytvoří se `*.c` C soubor, v něm se implementuje kód a poté se přeloží. Výsledkem je `*.ko` soubor, který se do jádra zavede jako modul nástroji `insmod` či `modprobe`.

Bylo zmíněno, že komunikace procesů s jádrem je omezena a jedním z možných způsobů je systémové volání `ioctl`. Dalším řešením je technologie netlink. Ten vznikl původně pro síťové operace v uživatelském prostoru (řízení paketového filtru, ...) [9]. Velká výhoda netlinku je standardní socketové rozhraní v uživatelském prostoru. Díky tomu s ním lze pracovat stejně jako s hojně používaným síťovým socketem. Události v jádře (například přidání nové směrovací cesty do hlavní směrovací tabulky) se uživatelským procesům oznamují přednostně tímto způsobem. Na obrázku 2.8 je zobrazena architektura rozhraní netlink. Pro čtení zpráv z jádra stačí zavolat funkce `socket()` pro vytvoření socketu, `bind()` pro přiřazení adresy a poté volat funkci `recv()` pro příjem zpráv.

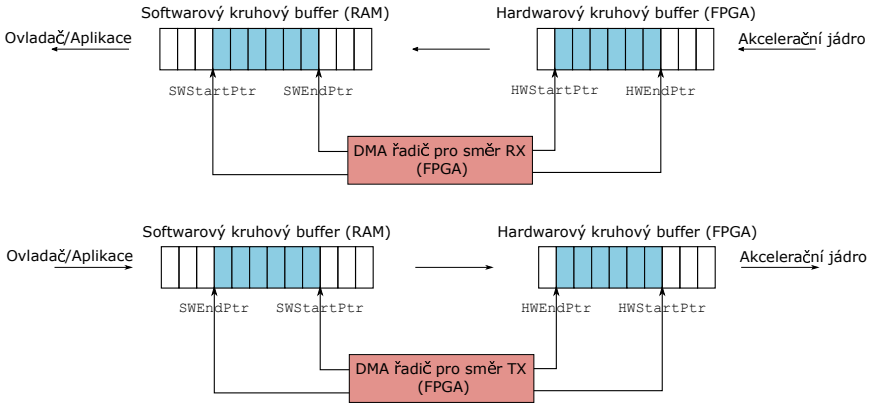


Obrázek 2.8: Architektura technologie netlink [9].

2.5 Rozhraní SZE2

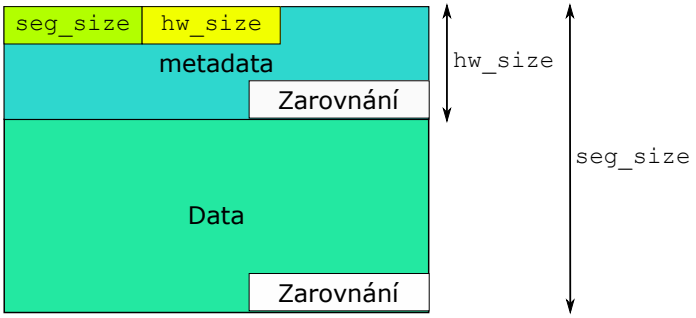
Podkapitola čerpá značnou část informací z diplomové práce Andreje Hanka [7] a bakalářské práce Matěje Vida [22]. SZE2 (Straight Zero Copy) je rozhraní pro rychlé DMA přenosy dat mezi akcelerační COMBO kartou a pamětí hostitelského systému. Na obrázku 2.9 je znázorněn princip přenosu dat přes rozhraní SZE2. Přenosy dat probíhají přes takzvané DMA kanály. Každý kanál je tvořen dvěma kruhovými buffery. Směr přenosu dat (příjem – RX/vyslání – TX) je oddělen – oba směry mají vlastní oddělené kruhové buffery. První kruhový buffer je umístěn v COMBO kartě, druhý se nachází v paměti hostitelského systému.

Každý kruhový buffer obsahuje 2 ukazatele: první na začátek dat `*PtrStart` a druhý na konec dat `*PtrEnd`. Vlastnost kruhový znamená, že ukazatel, který se dostane na hranici bufferu, je při dalším posunu vpřed přemístěn na začátek pole. Tím tak dochází k vytvoření pomyslného kruhu. Pomocí těchto ukazatelů se spravuje zaplnění a přenos dat mezi paměťmi. Přenos samotný je řešen hardwarovou částí při manipulaci s ukazateli. Přenos je realizován přes DMA kanály, kterých může být paralelně využito více najednou. Využitím např. 8 přijímacích DMA kanálů lze dosáhnout 8x vyšší propustnosti (je použito 16 kruhových bufferů). Velikost kruhového bufferu se pohybuje v řádech desítek až stovek MB.



Obrázek 2.9: Příjem dat nad rozhraním SZE2.

Data přenášená rozhraním SZE2 mají formu tzv. segmentů dat. Tyto segmenty mohou mít různou délku a menší segmenty lze agregovat do bloku dat, čímž se zvýší propustnost (díky snížení režie spojené s přenosy po sběrnici PCI-Express). Struktura segmentu je graficky znázorněna na obrázku 2.10. Segment se skládá z hlavičky a samotných dat. Hlavička obsahuje položky `seg_size`, `hw_size` a metadata. Položka `seg_size` má velikost 2 B a udává velikost celého segmentu (hlavička i data) v bajtech. Položka hw_size má taktéž 2 B a udává velikost hlavičky v bajtech. Maximální velikost segmentu je 64 kB (2^{16}). Samotná data typicky obsahují rámec z linkové vrstvy. Data i metadata musí být zarovnána na 8 B. Pokud nejsou, doplní se prostor nulami tak, aby zarovnána byla.



Obrázek 2.10: Struktura segmentu.

Softwarová část rozhraní SZE2 se nachází na hostitelském systému a je tvořena ovladači a uživatelskými knihovnami. Z uživatelských knihoven zde bude zmíněna pouze knihovna *libsze2*, která poskytuje nízkoúrovňové rozhraní k rychlým DMA přenosům – například umožňuje uživatelské aplikaci jednoduchou práci s jednotlivými rámci.

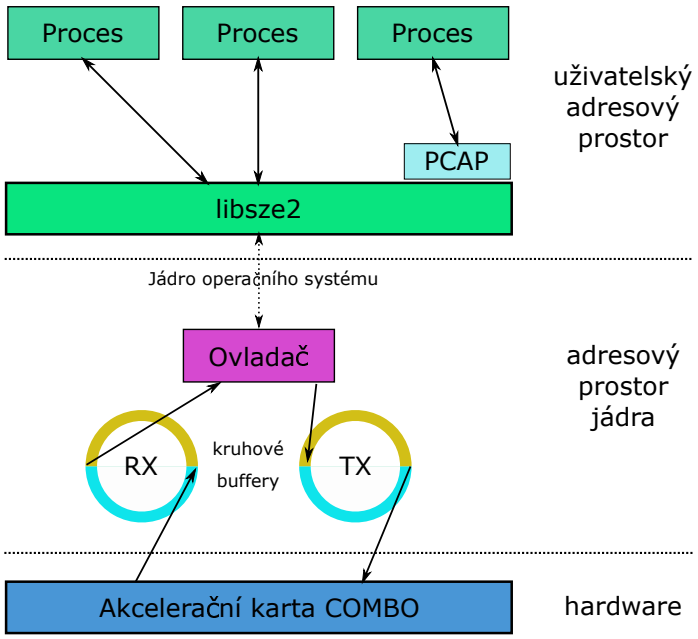
Ovladače umožňují komunikaci s hardwarem skrze jádro operačního systému ve formě modulů, které se mohou dynamicky vkládat a odebírat za běhu systému. Ovladače pro karty COMBO jsou napsány pro linuxové operační systémy. Mezi základní funkce ovladače patří:

- Inicializace a konfigurace karty.
- Správa DMA přenosů mezi hostitelským systémem a COMBO kartou.

Tuto funkcionalitu zajišťují následující ovladače:

- *combo** – tvoří nejnižší vrstvu, implementují operace pro konkrétní karty.
- *szedata2** – zajišťují operace pro rychlé DMA přenosy.

Ovladače szedata2 jsou rozděleny na hardwarově závislou a hardwarově nezávislou část. Hardwarově závislá část pracuje s ovladači nižší vrstvy COMBO a poskytuje operace nezávislé části. Hardwarově nezávislá část se v sytému zaregistruje jako ovladač znakového zařízení s cestou `/dev/szedataIIX`, kde X je číslo instance zařízení. Na obrázku 2.11 je



Obrázek 2.11: Princip komunikace mezi hardwarem a aplikacemi.

zobrazen princip komunikace uživatelských aplikací s kartou (hardwarem). Uživatelská aplikace využívá služeb knihovny *libsze2*. Ta vnitřně pracuje tak, že si otevře znakové zařízení `/dev/szedataIIX` a použitím systémových volání `ioctl` mu předává příkazy s parametry. Tyto příkazy jsou zachyceny ovladačem, který příkaz interpretuje a následně předá kartě příslušné instrukce.

Standardně se přenos dat mezi jádrem a uživatelskou aplikací řeší kopírováním dat. Jedná se ale o časově náročnou operaci a to je nežádoucí. Rozhraní SZE2 pro přenos síťových dat mezi aplikací a ovladačem (jádrem) využívá techniku mapování paměti. Ta umožňuje namapování paměti jádra do uživatelského prostoru. Aplikace v uživatelském režimu pak

může přímo přistupovat do paměti jádra, resp. ovladače. Výsledkem je značný nárůst výkonu. Aplikace ovladač požádá o namapování kruhového bufferu voláním standardní funkce znakového souboru `mmap`.

Vývoj COMBO karet probíhá nad platformou NetCOPE. Platforma NetCOPE je další z projektů Liberouteru a slouží pro rychlý a snadný vývoj síťových zařízení/aplikací využívajících akceleračních karet. Platforma NetCOPE vytváří nad kartou abstraktní vrstvu a umožňuje jednoduše využívat jejích zdrojů [7]. Skládá se z hardwarové i softwarové části. Výše popisované rozhraní SZE2 je její součástí – obsahuje tak mj. uživatelskou knihovnu *libsze2* nebo ovladače *combo** a *szedata2**. Hardwarová část zajišťuje rychlé DMA přenosy, síťové rozhraní pro vysílání a příjem (až 100 Gbps) a rozhraní pro komunikaci s hostitelským systémem (přes PCI-Express).

V průběhu této bakalářské práce vývojáři sdružení CESNET vytvořili nový softwarový balíček pro platformu NetCOPE s názvem *netcope-common*, který by měl postupně nahradit starší softwarovou část NetCOPE. Tento softwarový balíček zatím není zcela kompletní a je dále rozvíjen. Nad rámec zadání se tak tato práce bude zabývat nejen vytvořením ovladače pro standardní síťové rozhraní nad původním rozhraním SZE2, ale také nad novou softwarovou vrstvou – rozhraním NDP. Nový softwarový balíček se oproti starému liší především v těchto bodech:

- Podpora novějších linuxových jader od verze 3.x a výše.
- Softwarová podpora pro větší počet DMA kanálů (32+).
- Přehlednější kód a pozměněná struktura.
- Integrace podpory Device Tree (datová struktura popisující hardwarové komponenty systému) pro všechny softwarové části (ovladače, knihovny, nástroje). Nahrazuje tak soubor *design.xml*, který popis v minulosti řešil a musel být distribuován spolu s bitstreamem.
- Nová jména: SZE -> NDP[4], libsze2 -> libnfb[5].

Z pohledu ovladačů jsou staré moduly *combo** a *szedata2** sloučeny do jednoho nového modulu s názvem *nfb*. Samotný princip DMA přenosů, jako je formát segmentu a způsob přenosu dat pomocí dvojice kruhových bufferů zůstává zcela zachován. Mění se pouze API (aplikační rozhraní), které nově poskytuje daleko vyšší flexibilitu a je daleko více přímočaré při použití ze strany uživatelské aplikace.

[4] NetCOPE Data Plane
[5] NetCOPE FPGA Board

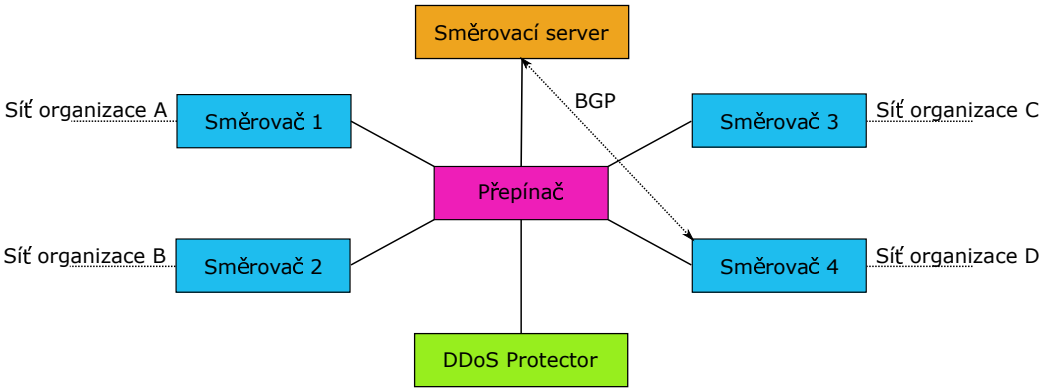
Kapitola 3

Konceptuální návrh systému, architektury ovladače

3.1 Zapojení DDoS Protectoru, příjem a odesílání dat

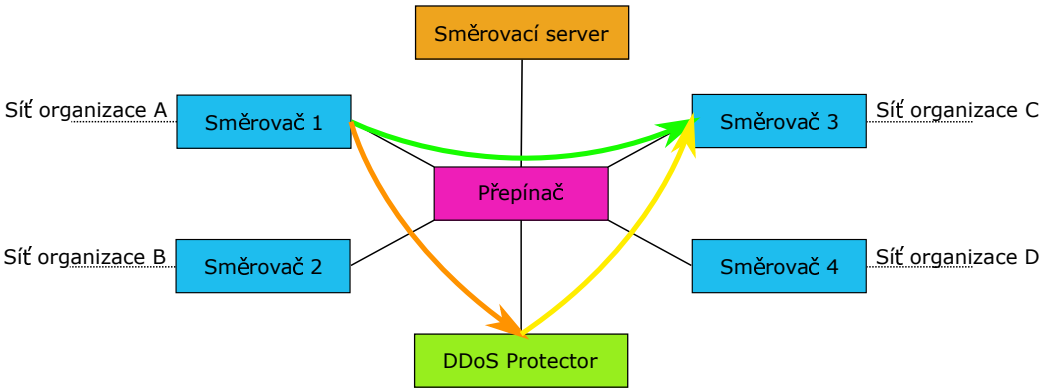
Zařízení DDoS Protector se musí v síti chovat jako standardní síťové zařízení. To zahrnuje funkcionalitu definovanou vrstvami TCP/IP modelu, tzn. musí podporovat standardní síťové protokoly. V případě Protectoru se jedná konkrétně o protokoly ARP, NDP, DHCP, ICMP a BGP. Na obrázku 3.1 je znázorněn alternativní způsob zapojení Protectoru do peeringového uzlu v rámci L2 infrastruktury. Peeringový uzel je označení pro vzájemné propojení počítačových sítí více telekomunikačních společností s cílem výměny datového provozu. Jednotlivé peeringové uzly tvoří vlastně páteř celého Internetu. Jedním z českých peeringových uzlů je například sdružení NIX.CZ [16]. Na rozdíl od zapojení popsaného v podkapitole 2.3 není Protector připojen do sítě prostřednictvím směrovače, který by přímo zajišťoval směrování podezřelého provozu do Protectoru a návrat odfiltrovaného provozu zpět do cílové sítě. Protector tak musí v tomto případě zajišťovat nejen filtraci, ale částečně i směrování provozu. Uprostřed infrastruktury sítě peeringového uzlu se nachází výkonný přepínač (jak je znázorněno na obrázku 3.1), resp. struktura více přepínačů (z důvodu zajištění redundance a požadované spolehlivosti). Do něj jsou připojeny směrovače jednotlivých organizací připojených do peeringového uzlu. Do přepínače je také připojen Protector a směrovací server, resp. více směrovacích serverů, opět z důvodu redundance. Směrovače připojených organizací komunikují pomocí protokolu BGP (znázorněno přerušovanou čarou) se směrovacím serverem a oznamují přes něj, které cílové IP prefixy přes ně mají být směrované.

Na obrázku 3.2 je znázorněna situace přesměrování podezřelého síťového provozu přes Protector a navrácení odfiltrovaného provozu zpět do cílové sítě. Zelenou křivkou je zobrazena výchozí cesta provozu z organizace A směřujícího do sítě organizace C. V situaci, kdy je na cílový prefix organizace C veden útok a daná koncová síť chce využít Protector pro filtraci provozu, je z dané sítě vyslána BGP zpráva na směrovací server. Ta nese informaci, na základě které se na směrovacím serveru přepíše pravidlo určující next-hop do zahlcené sítě a to ze směrovače 3 na Protector, zobrazené oranžovou křivkou. Směrovací server tuto informaci oznámí dalším směrovačům. Next hop je IP adresa, na kterou jsou směrovány pakety pro daný cílový prefix. To znamená, že Protector musí mít nastavenu IP adresu. V dnešních sítích jsou IP adresy často přidělovány pomocí DHCP protokolu, proto je vhodné, aby Protector uměl komunikovat pomocí DHCP protokolu a uměl si takto při



Obrázek 3.1: Zapojení protectoru v L2 infrastruktuře peeringového uzlu.

připojení do sítě automaticky získat vlastní IP adresu. Směrovač, který má nyní posílat provoz na Protector zná jeho IP adresu, ale pro odeslání musí ještě zjistit MAC adresu jeho ethernetového rozhraní. Směrovač vyšle všesměrový (broadcast) ARP dotaz na MAC adresu Protectoru a to všem zařízením připojeným do této sítě. Aby Protector mohl odpovědět, tak musí podporovat protokol ARP. V případě IPv6 se nepoužívá ARP ale protokol NDP. Nyní se už provoz dostane do Protectoru, ten jej vyfiltruje a následně jej potřebuje odeslat do cílové sítě, na obrázku 3.2 zobrazeno žlutou křivkou. K tomu je nutné znát next-hop, který je uložen na směrovacím serveru. Protector tedy musí umět komunikovat přes BGP, stejně jako ostatní směrovače. Až zjistí IP adresu next-hopu, musí ještě vyhledat její MAC adresu, pokud ji už nemá v ARP cache paměti. Musí tak odeslat ARP nebo NDP dotaz. Teprve poté může paket odeslat.



Obrázek 3.2: Změna cesty nebezpečného provozu přes Protector.

Protector využívá akcelerační kartu COMBO pro urychlení výpočetních operací. Pro velmi rychlý přenos dat mezi kartou COMBO a operační pamětí Protectoru se využívá rozhraní SZE2. To využívá mechanismu DMA, kdy se vytvoří několik párů DMA kanálů (směr RX a TX), na nichž pak paralelně dochází k přenosu dat oběma směry. Uživatelská aplikace získá přístup do paměti jádra (ovladače) ke čtení/zápisu kruhových bufferů díky technice mapování paměti, čímž kompletně obchází síťovou vrstvu jádra. Pro získání funkcionality požadovaných protokolů se tak nabízejí dvě možnosti:

Implementovat protokoly ve vlastní režii nad rozhraním SZE2 Řešení by bylo implementačně velmi náročné a to z několika důvodů. Nyní je potřeba podporovat pět protokolů, ale je možné, že v budoucnosti tento počet vzroste. Jedná se tak o ztrátu určité flexibility, kdy by se s každým dalším požadavkem musel protokol implementovat ručně. Implementace samotného protokolu také nemusí být jednoduchá, například RFC dokument popisující chování BGP protokolu obsahuje 104 stran [20]. Protokoly jako DHCP nebo BGP pracují na L7 vrstvě, tzn. příchozí rámec musí projít síťovou vrstvou skládající se z TCP/IP zásobníku, který by se také musel implementovat. To zahrnuje například vlastní správu ARP cache na L2/L3 vrstvě, řešení fragmentace a znovusestavení (reassembling) paketu na L3 vrstvě nebo navázání a ukončení TCP spojení na L4 vrstvě. Výhoda tohoto řešení by byla možnost optimalizovat implementaci na rychlost, kdy by nedocházelo ke kopírování dat, ať už mezi jádrem a uživatelským procesem (technika mapování paměti) nebo přímo v jádře, kdy by se data nepředala síťové vrstvě jádra.

Vytvořit ovladač síťového zařízení a využít tak podpory v jádře Toto řešení je mnohem méně implementačně náročné. Je flexibilní, protože jádro již obsahuje všechny standardní síťové protokoly a navíc se neustále vyvíjí, takže nebude potřeba se zabývat aktualizacemi stávajících protokolů a stejně tak se do jádra dostanou i nové síťové protokoly. Nevýhoda je vyšší dopad na celkový výkon Protectoru kvůli kopírování dat. Linuxové systémy obsahují démona, který řeší protokol DHCP, ale ten jakožto L7 protokol není implementován v jádře. BGP protokol standardně podporován není. Možné řešení by bylo použít směrovacího démona BIRD, který by naslouchal a odpovídal na vytvořeném síťovém rozhraní. Příklad demonstrující rozdíl mezi L7 a L4/3/2 protokoly: Pokud by rozhraní přijmulo ARP dotaz na jeho MAC adresu, tak by tento dotaz odchytilo samo jádro a zajistilo by odpověď. Pokud by rozhraní přijmulo BGP zprávu, jádro by tato „neznámá“ data předalo procesu s odpovídajícím portem, což by byl BIRD.

Z pohledu Protectoru je přitom nutné rozlišovat provoz na ten, který je určen přímo pro Protector (jakožto koncové zařízení) a ten, který přes Protector pouze prochází a je současně filtrován. U filtrovaného provozu je potřeba vysoký výkon a řídící aplikace k němu přistupuje přes SZE2 rozhraní. U provozu směrovaného přímo na Protector (cílová IP adresa odpovídá IP adrese Protectoru) naopak není nutné disponovat vysokým výkonem. Jedná se o již zmíněné ARP dotazy/odpovědi, BGP komunikaci či spojení SSH. Bez ohledu na zvolené řešení podpory standardních síťových protokolů bude nutné vyhradit jeden RX DMA kanál pro příjem paketů a jeden TX DMA kanál pro odeslání paketů, tedy dohromady alespoň 2 DMA kanály. To se projeví na propustnosti Protectoru, protože tyto kanály již nebudou využity pro filtrovaný provoz, ale pro provoz směřovaný na Protector. Ve směru příjmu je totiž nutné vědět, kde se síťová data nachází, resp. v jakém kruhovém bufferu, aby s nimi bylo možné dále pracovat. Protector může mít odlišné DMA kanály mapované na odlišná fyzická rozhraní, tudíž i pro směr odesílání je nutné vědět, do kterého kruhového bufferu data zapsat tak, aby například odpověď Echo Reply na ICMP zprávu Echo Request byla odeslána na stejné fyzické rozhraní, ze kterého přišel požadavek. Pro zajištění výkonu Protector standardně multiplexuje provoz na všechny DMA kanály rovnoměrně. Je potřeba zajistit, aby provoz směřovaný přímo pro Protector šel přes vybrané DMA kanály, zatímco zbytek provozu přes zbytek DMA kanálů. To lze zajistit nastavením hardwarových filtrů ve firmware. Jednoduchý příklad takového filtru by spočíval v kontrole cílové IP adresy. Pokud by se cílová IP adresa přijatého rámce shodovala s IP adresou fyzického rozhraní

karty, byl by rámec poslán na vybraný(é) DMA kanál(y). Kromě cílové IP adresy by se měly kontrolovat i speciální cílové MAC adresy, jako jsou broadcast (FF:FF:FF:FF:FF:FF) nebo multicast.

Zadání práce přímo specifikuje implementaci síťového ovladače, nicméně z analýzy obou možností pro podporu požadovaných protokolů i tak vyplývá, že takové řešení je pro Protector výrazně výhodnější. Jeho slabina, nižší celková propustnost, je zmírněna skutečností, že pro daný provoz není nutné disponovat vysokým výkonem. Zároveň je získána velká flexibilita, což může být značná výhoda do budoucna, jelikož se Protector se neustále vyvíjí a nelze vyloučit vznik nových požadavků na podporu jiných síťových protokolů.

3.2 Architektura ovladače

Následující podkapitola se věnuje přímo samotnému návrhu architektury ovladače. Ovladač síťového zařízení bude muset provádět řadu důležitých operací, mezi které patří:

Inicializace. Kromě klasických inicializačních úkonů jako alokace paměti nebo inicializace struktur bude nutné v jádru zaregistrovat síťové zařízení `netdev`. Podle konfigurace ovladače se může zaregistrovat i více síťových zařízení odpovídajících různým DMA kanálům, případně různým fyzickým rozhraním přítomným na akcelerační kartě.

Otevření síťového rozhraní. Zde se bude třeba přes vnitřní struktury SZE2 přihlásit k odběru DMA kanálů – tak, aby bylo zřejmé, že existuje nějaký proces, který bude na těchto DMA kanálech číst nebo zapisovat data. Akcelerační karta pak bude autonomně přenášet síťová data do softwarového kruhového bufferu a naopak. Mapování DMA kanálů na síťové rozhraní ale nemusí být úplně jednoznačné. Pro příjem lze využívat určitý počet RX DMA kanálů, naopak pro odesílání zase jiný počet TX DMA kanálů. Jednoduchý přístup by spočíval ve vytvoření síťového rozhraní pro každý RX/TX DMA pár (viz Inicializace). Otevření síťového rozhraní 0 by pak znamenalo přihlášení k odběru TX DMA kanálu 0 a RX DMA kanálu 0.

Příjem paketů. Ve standardním řešení příjmu paketů se využívá přerušení, kdy se pro každý paket vykoná obslužná funkce. Existuje ještě druhý způsob – polling, který ve vhodný čas zkontroluje stav síťové karty a pokud jsou k dispozici nová data, tak je zpracuje. Polling má vyšší výkon oproti technice přerušení [9] a tak je účelnější jej využít, pokud se jedná o ovladač nad vysokorychlostní síťovou kartou. Jádro nabízí možnost využít polling správným nastavením `netdev`u, ale v tomto případě je navrženo vlastní daleko jednodušší řešení. To spočívá ve formě vytvoření jaderného vlákna, které bude v pravidelném intervalu kontrolovat, zda nejsou dostupná nová data (přes ukazatele kruhových bufferů). Vytvoření vlákna tedy bude nutné provést již při otevření rozhraní. Pokud bude otevřeno více síťových zařízení, bude vytvořeno i více vláken, díky čemuž lze dosáhnout zlepšení výkonu. Samotný příjem paketů se bude skládat z načtení paketu z kruhového bufferu a následného předání paketu síťové vrstvě jádra. Protože jsou pakety procházející rozhraním SZE2 zabaleny do segmentů, bude nejdříve nutné odstranit hlavičku.

Odesílání paketů. Odesílání paketů by se řešilo standardním způsobem, kdy se pro každý paket zavolá funkce pro odeslání. Odesílání bude fungovat obdobně jako u příjmu, tzn. odesílaný paket se vloží do kruhového bufferu a následně se kartě vydá pokyn k odeslání. Paket se bude muset nejprve zabalit do segmentu. Stejně jako u příjmu není

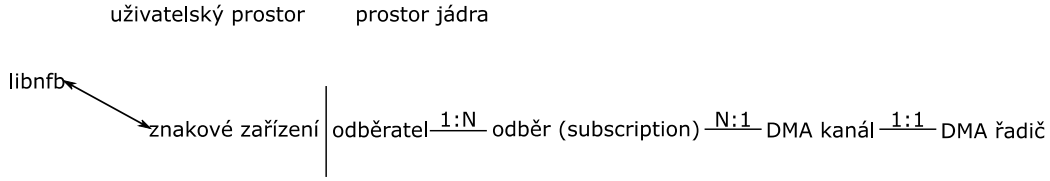
SZE2 hlavička potřeba a její velikost se nastaví na nula bajtů. Kromě velikosti hlavičky se musí nastavit i velikost celého segmentu (SZE hlavička + data paketu). V takovém případě je potřeba dávat pozor na minimální velikost Ethernetového rámce, která činí 64 bajtů. Akcelerační karta totiž neumožňuje přímé odesílání rámců kratších, než dovoluje standard rozhraní Ethernet.

Zavření síťového rozhraní. Dojde k odhlášení všech DMA kanálů, ke kterým bylo síťové rozhraní přihlášené a ukončí se příslušné jaderné vlákno pro příjem.

Deinicializace. Provede odregistraci síťového zařízení a uvolní alokované zdroje.

Ostatní podpůrné operace. Patří sem například možnost libovolně nastavit MAC adresu. Kontrolu MAC adresy lze nastavit již na COMBO kartě, ale v případě potřeby je možné MAC adresu jejího fyzického rozhraní změnit a v takovém případě se musí změnit i MAC adresa síťového zařízení, jinak by jádro paket zahodilo. Při příjmu rámců síťovou vrstvou jádra se totiž provádí také kontrola MAC adres. Další operací je sběr provozních statistik v podobě počtu přijatých bajtů, paketů a počtu odeslaných bajtů, paketů. To zahrnuje nutnost inkrementovat příslušné čítače u operací pro příjem a odeslání paketů.

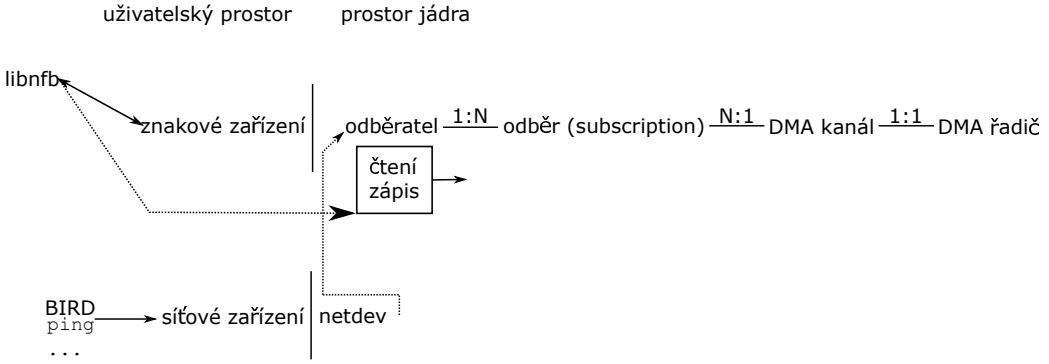
Na obrázku 3.3 se nachází struktura NDP rozhraní nových ovladačů. Ta se přitom příliš neliší od struktury staršího rozhraní SZE2. Ovladač síťového rozhraní se v případě SZE2 pouze zařadí do modulu hardwarově nezávislé části. Uživatelská aplikace využívající *libndp* knihovnu je v ovladači vnitřně reprezentována strukturou `subscriber` (odběratel). Ten může požádat o přístup k jednotlivým DMA kanálům. Tato žádost o přístup je reprezentována strukturou `subscription` (odběr). Jak vztahy na obrázku značí, 1 odběratel může mít přístup k více DMA kanálům, 1 DMA kanál může mít více odběratelů (například když více ladících nástrojů potřebuje přístup k samému DMA kanálů).



Obrázek 3.3: Struktura nových NDP ovladačů.

Podpora NDP přenosů do uživatelského prostoru je kompletně implementovaná a uživatelské procesy tak mohou pracovat bez potíží stejně jako u starších ovladačů. Uživatelská knihovna *libndp* obsahuje mj. funkce pro jednoduchý příjem a odesílání paketů a tyto funkce komunikují s ovladači NDP rozhraní přes `ioctl` volání, které obsahují pokyny typu přihlášení k odběru DMA kanálu nebo nastavení ukazatelů kruhových bufferů. Z pohledu ovladače síťového zařízení nad novými NDP ovladači by bylo vhodnější využít již naimplementované a otestované funkcionality pro příjem a odesílání. Na obrázku 3.4 je zobrazen návrh na začlenění ovladače síťového zařízení do nových ovladačů NDP rozhraní. K tomu, aby bylo možné použít funkce z *libndp* knihovny v ovladači, tzn. v režimu jádra, bude nutné funkce implementované pro uživatelský režim patřičně upravit. Jednou z nutných úprav bude překlenutí `ioctl` volání a použít přímo ty funkce, které by byly při daném `ioctl` volání provedeny. Takto upravené funkce by se měly přidat k ostatním obecným NDP funkcím (jako třeba

práce s kruhovými buffery, správa odběrů, ...) a měly by tak být všeobecně využitelné. Ovladač síťového zařízení pak může využít tyto jednoduché funkce pro příjem a odesílání nad přihlášenými DMA kanály. Návrh jinak zůstává stejný jako u ovladače pro SZE2 rozhraní.



Obrázek 3.4: Návrh na přidání netdevu do nových ovladačů.

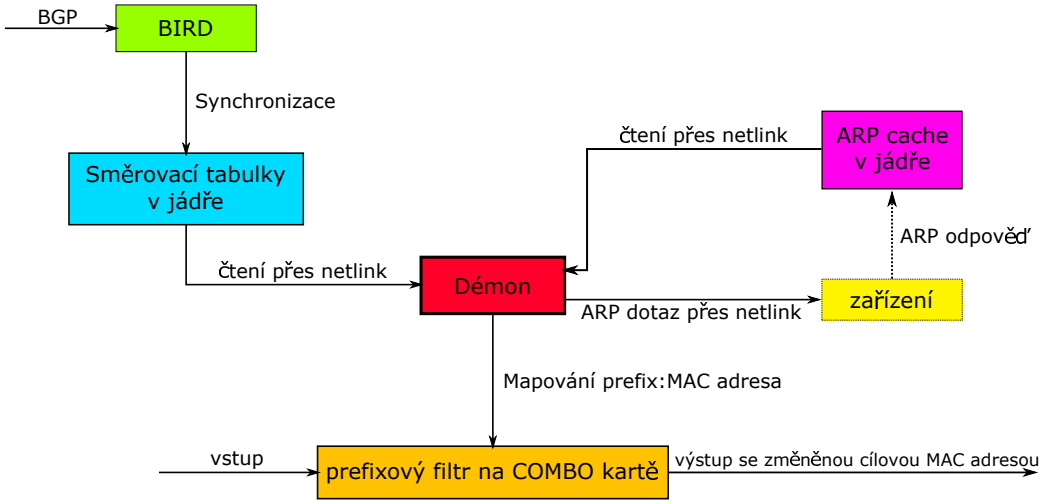
3.3 Akcelerované přeposílání paketů

Samotné směrování paketů vyfiltrovaného provozu musí probíhat na vysokých rychlostech. Firmware COMBO karty využívaný Protectorem proto obsahuje také hardwarovou akceleraci takovéto operace. Podpora spočívá v přítomnosti specializovaného prefixového filtru, který obsahuje informace o mapování cílového IP prefixu na MAC adresu next-hopu. Na základě vyhledání v tomto filtru následně hardware provádí přímý přepis cílové MAC adresy na vyhledanou hodnotu. Informace o mapování je ale potřeba do filtru dynamicky nahrávat (upravovat) na základě aktualizací směrovacích informací získaných prostřednictvím BGP protokolu. Z toho důvodu byla nad rámec zadání navržena uživatelská aplikace, přesněji démon, který pracuje na pozadí a provádí prvotní inicializaci a pravidelnou periodickou aktualizaci těchto informací v hardware.

Pro čtení směrovacích informací BGP protokolu lze, jak už bylo zmíněno v předchozích podkapitolách, využít směrovacího démona BIRD. Ten interně pracuje s vlastními směrovacími tabulkami. Každý podporovaný protokol je možné různě nakonfigurovat. U BGP lze nastavit například číslo autonomního systému, ve kterém se nacházíme, nebo nastavit, kterou vlastní směrovací tabulku má aktualizovat v případě nové směrovací informace. BIRD umí přijímat, zpracovat a odpovídat na zprávy směrovacích protokolů, ale sám už neprovádí aktuální přeposílání paketů.

BIRD je open-source software a první možností, jak zajistit aktualizaci hardwarového prefixového filtru na COMBO kartě, je upravit jeho zdrojové kódy a směrovací informace z interních tabulek BIRD přímo synchronizovat s pravidly prefixového filtru. Po přijmu nové BGP zprávy by byl znám cílový prefix i IP adresa next-hopu, ale také bylo by nutné dodatečně vyhledat MAC adresu. MAC adresy jsou uloženy v systémové ARP cache tabulce, a pokud by se v ní záznam pro danou IP adresu nenacházel, musel by se zjistit ARP dotazem. Vhodným kandidátem pro čtení ARP cache i odeslání ARP dotazu je technologie netlink. Nahrání informace o mapování do prefixového filtru by poté proběhlo přes knihovnu *libdcpro*, která slouží jako softwarové API pro komunikaci s firmwarem karty. Druhou variantou je možnost využít synchronizace směrovacích informací z interních tabulek BIRD do systémové směrovací tabulky linuxového jádra a vytvořit vlastní aplikaci, která by četla informace

o cestách ze systémové směrovací tabulky a to opět pomocí technologie netlink. Získání MAC adresy next-hopu by bylo totožné s první možností. Z pohledu návrhu je tato volba čistější řešení, jelikož celá aplikace bude postavena nad prací s netlinkem, tzn. nedochází k propojení odlišných technologií. Dále se jedná o klasickou aplikaci, tudíž bude volnější možnost implementace, oproti nutnosti program vhodným způsobem zakomponovat do již hotového kódu BIRD. Třetí výhodou je flexibilita, kdy aplikaci bude možné dále využívat i při změně směrovacího démona (za předpokladu, že nový směrovací démon bude umět plnit systémovou směrovací tabulku) a také například možnost manuálně přidat směrovací cesty standardními linuxovými nástroji (`ip route add...`). Na obrázku 3.5 je zobrazeno zapojení aplikace v systému. BIRD (vlevo nahoře) přijímá směrovací cesty přes protokol BGP a zapisuje je do systémové směrovací tabulky. Změny v této tabulce sleduje démon prostřednictvím technologie netlink. V případě nového záznamu o cestě se, pokud ještě není známa, vyhledá MAC adresa next-hop směrovače (žlutá barva) odesláním ARP dotazu skrze netlink. Odpověď odchytí jádro a přidá nový záznam do systémové ARP cache, u které démon taktéž sleduje změny. Když už je znám cílový prefix i MAC adresa next-hopu, je tento záznam nahrán do hardwarového prefixového filtru (oranžová barva).



Obrázek 3.5: Návrh aplikace pro nahrávání informací do hardwarového prefixového filtru.

Jak již bylo uvedeno, BIRD neprovádí aktuální přeposílání paketů, ale umožňuje se-synchronizovat interní tabulky se směrovacími tabulkami v jádře, které na základě těchto tabulek už přeposílání provádí. Konfigurace BIRDa pro provádění takové synchronizoval je poměrně jednoduchá (je to také jeden z cílů autorů [2]) a jedná se o jeho standardní součást. Následuje ukázka jednoduché konfigurace s komentáři, která přijímá BGP informace a ty synchronizuje se systémovou směrovací tabulkou číslo 10:

```
table DDOS;                # definuj vlastni smerovaci tabulku DDOS

protocol kernel k_DDOS {   # specialni protokol "kernel" pro synchronizaci
                           # se systemovymi smerovacimi tabulkami

   table DDOS;             # synchronizuj interni tabulku DDOS se ...
   kernel table 10;        # ... systemovou smerovaci tabulkou 10
```

```
   export all;            # synchronizace je provedena tak, ze obsah
                          # tabulky DDOS je vlozen do tabulky 10

   import none;           # ale naopak do tabulky DDOS nic nevkladej
                          # jde tak o jednocestnou synchronizaci

   persist on;            # data v tabulce 10 zustanou i po ukonceni BIRDa
}

protocol bgp BGP1 {       # konfigurace BGP
   local as 65506;        # cislo naseho autonomniho systemu
   table DDOS;            # BGP informace aktualizuj s tabulkou DDOS
   import all;            # prijmi vsechny BGP zpravy ktere dostanes
   export none;           # sam ale zadne BGP informace nebudes posilat
   neighbor 2001:7f8::1 as 60143;   # sousedici smerovac, se kterym
                                    # bude probihat komunikace
}
```

Kapitola 4

Implementace

Kapitola se věnuje implementaci a jejím vybraným detailům. Podkapitola 4.1 popisuje implementaci ovladače síťového zařízení nad rozhraním SZE2, zahrnujíc existující adresářovou strukturu ovladačů, její popis a některé detaily implementace. Navazující podkapitola 4.2 popisuje totéž nad novějším rozhraním NDP. Podkapitola 4.3 se zabývá implementací démona pro zapisování pravidel do hardwarového prefixového filtr.

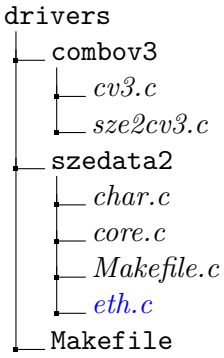
4.1 Rozhraní SZE

Nad starší generací karet COMBO historicky fungovala implementace ovladače síťového zařízení realizující požadovanou funkcionalitu. Od té doby se ale zásadním způsobem změnil způsob příjmu paketů, kdy novější generace karet negeneruje přerušení při příchodu nových dat. Při implementaci ovladače tak bylo vycházeno z existující starší verze, avšak nutně byly provedeny nezbytné úpravy pro současný způsob příjmu paketů přes rozhraní SZE2 u novějších karet COMBO.

Na obrázku 4.1 je zobrazena zjednodušená adresářová struktura (uvedeny jsou pouze relevantní soubory a adresáře) současných ovladačů COMBO. Adresář combov3 tvoří nejnižší vrstvu a implementuje operace pro třetí generaci karet COMBO. Patří sem i hardwarově závislá část rozhraní SZE2 (správa komunikace přes DMA, ...). Adresář szedata2 implementuje hardwarově nezávislou část rozhraní SZE2. Jedná se o část, kam by se měl zařadit i vyvíjený ovladač síťového zařízení. Soubor *char.c* zajišťuje operace znakového zařízení `/dev/szedataIIX`, jako například otevření zařízení nebo systémové volání `ioctl`. Soubor *core.c* poskytuje základní operace jako je alokace struktur, přihlášení k odběru DMA kanálů nebo čtení a zápis z/do DMA kanálů. Samotný ovladač je reprezentován souborem *eth.c*, jež je označen modrou barvou. Ovladač není z důvodu již zavedených souvislostí vytvořen jako samostatný jaderný modul, ale jako součást modulu *szedata2.ko*, který vznikne spojením souborů ve stejnojmenném adresáři. Hardwarově nezávislý modul *szedata2.ko* se zavede jako první a připraví operace pro hardwarově závislý SZE2 modul *szedata2_cv3.ko*, který následně vytvoří a zaregistruje znakového zařízení `/dev/szedataIIX`. Spolu s tím zároveň provede inicializaci ovladače síťového zařízení.

Následuje popis vybraných implementačních detailů z operací, které byly navrženy v rámci podkapitoly 3.2:

Inicializace. Tato operace získá počet RX a TX DMA kanálů jako parametr. V této fázi se provádí jednoduchý způsob vytvoření jednoho síťového zařízení pro každý pár TX/RX DMA kanálů. Poté se vytvoří a v systému zaregistruje síťové rozhraní pod



Obrázek 4.1: Zkrácená struktura COMBO ovladačů.

jménem `cXethY`, kde X je číslo instance znakového zařízení szedataIIX a Y je číslo páru TX/RX DMA kanálů začínající od 0. MAC adresa takto nově vytvořeného zařízení je vygenerována náhodně, avšak první polovina MAC adresy je fixní a odpovídá zaregistrované hodnotě sdružení CESNET. Tato hodnota činí 00:11:17 [8].

Otevření síťového rozhraní. Vychází se z návrhu a obrázku 3.3, kdy je vytvořen jeden odběratel, v SZE2 označen jako aplikace (app) reprezentující dané síťové zařízení. Ten se přihlásí k odběru (subscription) TX a RX DMA kanálů o indexu Y (viz předchozí bod Inicializace). Pak je vytvořeno jaderné vlákno pro příjem, které je následně nutné probudit.

Příjem paketů. Tato operace využívá funkci `szedata2_eth_rx_data()` pro čtení paketů, kdy funkce naplní strukturu `sze2_adesc` počtem přečtených bajtů a pozicí (offsetu) ukazatele v kruhovém bufferu na začátek dat. Uvnitř cyklu se poté zpracovávají jednotlivé pakety. Rozhraní SZE2 přenáší data v segmentech, tzn. obsahují SZE2 hlavičku (obrázek 2.10) a poté samotná data paketu. Zpracování probíhá tak, že se nejdříve přečtou první 4 B segmentu, z nichž se zjistí velikost hlavičky a velikost samotného paketu. Protože jsou segmenty naskládány lineárně za sebou, stačí k získání pozice dat paketu v kruhovém bufferu přičíst k (lokálnímu) offsetu velikost SZE2 hlavičky. Pak se postupuje standardním způsobem, tzn. alokuje se paměť pro strukturu `sk_buff`, do ní jsou z kruhového bufferu nakopírovány samotná data paketu (tedy bez SZE2 hlavičky – ta byla odstraněna přičtením její velikosti k offsetu) a paket je předán síťové vrstvě jádra. Zaktualizují se čítače pro počet přenesených paketů a bajtů. K (lokálnímu) offsetu je přičtena velikost právě zpracovaného paketu, aby ukazoval na další segment v kruhovém bufferu a cyklus se opakuje. Po zpracování všech přečtených paketů je nakonec volána funkce `szedata2_rxunlock_data()`, která provede synchronizaci ukazatelů do kruhových bufferů a uvolní tak místo po všech přečtených paketech. Tento proces se neustále opakuje ve smyčce. V případě, že nejsou žádná nová data k dispozici (funkce `szedata2_eth_rx_data()` přečte nula bajtů), je vlákno uspáno na 1 ms, aby se ušetřil čas procesoru.

Odesílání paketů. Tato operace využívá funkci `szedata2_txlock_data()`, která připraví místo v kruhovém bufferu daného TX DMA kanálu. Jako jeden z parametrů se jí předává počet bajtů, které budou zapsány. Tento počet se skládá z velikosti odesílaného paketu a SZE2 hlavičky. SZE2 hlavička není v případě odesílání dat využita a její velikost je tak nulová (resp. nejsou využity její metadata, položky `seg_size` a `hw_size`

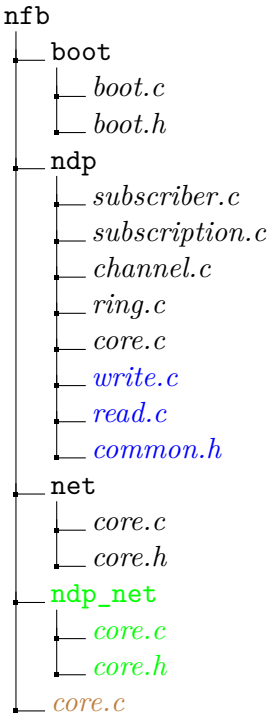
jsou povinné a je nutné s nimi vždy počítat, proto reálná velikost hlavičky je 8 B – 4 B zabírají položky `seg_size` a `hw_size`, ale data musí být zarovnána na 8 B). Délka samotného paketu se dá zjistit ze struktury `sk_buff`. Je však potřeba zkontrolovat, že délka odesílaného paketu je větší než minimální povolená délka ethernetového rámce. Ta činí 64 B s tím, že poslední 4 B – FCS vyplňuje přímo firmware karty a nutná minimální velikost tak činí 60 B. Například ARP dotaz má délku menší (46 B [17]) a tak se musí v segmentu vyplnit větší délka a paket musí být na minimální délku rozšířen. V jádru je pro toto porovnání definována konstanta `ETH_ZLEN`, kterou lze využít. Přes strukturu `sze2_adesc` se zjistí poloha ukazatele do kruhového bufferu (offsetu) a poté je možné zapsat 8 B SZE2 hlavičky a samotná data paketu. V případě paketů menších než konstanta `ETH_ZLEN` je zbylý prostor vyplněn nulami. Zaktualizují se čítače pro počet přenesených paketů a bajtů a uvolní se struktura `sk_buff`. Nakonec volána funkce `szedata2_txunlock_data()`, která provede synchronizaci ukazatelů do kruhových bufferů a tím způsobí samotné odeslání paketu z karty.

4.2 Rozhraní NDP

Na rozdíl od starších ovladačů SZE2 jsou nové NFB ovladače navrženy jako jediný jaderný modul `nfb.ko`. Ten obsahuje vlastní řešení určité podoby podmodulů, které jsou po zavedení vlastního modulu (`modprobe nfb.ko`) automaticky také připojeny. Na obrázku 4.2 je zobrazena zjednodušená struktura nové podoby ovladačů. Důležitým souborem je `core.c` v hlavním adresáři *kernel/driver/nfb*, který obsahuje inicializační a deinicializační funkce modulu jako celku. V rámci zavedení modulu se pak právě zde volají inicializační funkce jednotlivých podmodulů z adresářů *boot*, *ndp* a *net*, které musí tyto funkce implementovat. Pro přidání ovladače síťového zařízení pracujícího nad NDP DMA kanály tak bylo nutné tento soubor upravit a přidat napojení a odpojení nového submodulu. V rámci adresářové hierarchie na obrázku 4.2 jsou nově přidané soubory nového submodulu označeny zeleně. Je vytvořen nový adresář *ndp_net* a v něm se vytvoří dva soubory – *core.c* obsahující kód a hlavičkový soubor *core.h* s prototypy funkcí a použitými strukturami. Struktura je podobná submodulu *net*. Ten v nových ovladačích také vytváří ovladač síťového rozhraní, ale pouze pro účely čtení statistik z karty. Podle podkapitoly 3.2 bylo při implementaci využito již existujícího kódu pro odesílání a příjem, který se nachází v uživatelské knihovně *libnfb*. Ten byl upraven tak, aby ho bylo možné použít v režimu jádra. Výsledné soubory jsou zobrazeny modrou barvou, kde do adresáře NDP jsou přidány soubory *read.c* realizující příjem, *write.c* realizující odesílání a hlavičkový soubor *common.h* obsahující mj. definici používaných struktur. Úprava těchto souborů spočívala zejména v překlenutí volání `ioctl`, která se ve funkcích vyskytovala a byla nahrazena kódem, který se v ovladači při daném volání vykoná. U takového kódu bylo odstraněno kopírování dat z uživatelského prostoru do prostoru jádra a naopak.

Následuje popis vybraných implementačních detailů z operací, které byly navrženy v rámci podkapitoly 3.2:

Inicializace. Jedná o funkci, která se zavolá při zavedení tohoto submodulu. Z nového popisu komponent Device Tree je nutné nejdříve vyčíst, kolik DMA kanálů firmware karty obsahuje. Stejně jako u ovladače nad rozhraním SZE2 byl implementován způsob vytvoření jednoho síťového zařízení pro každý pár TX/RX DMA kanálů. Zkontroluje se tak, že počet RX a TX DMA kanálů je stejný a poté se pro každý pár vytvoří a v systému zaregistruje síťové rozhraní pod jménem `nfbXdY`, kde X je číslo instance



Obrázek 4.2: Zkrácená struktura nových NFB ovladačů.

znakového zařízení nfbX a Y je číslo páru TX/RX DMA kanálů začínající od 0. MAC adresa takto nově vytvořeného zařízení je vygenerována stejným způsobem jako u ovladače nad rozhraním SZE2.

Otevření síťového rozhraní. Vychází se z návrhu a obrázku 3.4, kdy je vytvořen jeden odběratel (subscriber) reprezentující dané síťové zařízení. Ten se přihlásí k odběru (subscription) TX a RX DMA kanálů o indexu Y (viz předchozí bod Inicializace). Poté je vytvořeno jaderné vlákno pro příjem, které je následně nutné probudit.

Příjem paketů. Tato operace využívá upravenou funkci pro příjem:

```
int kndp_rx_burst_get(struct ndp_queue *q, struct ndp_packet *packets,
                      unsigned count, struct ndp_subscriber *suber)
```

Parametr `suber` obsahuje ukazatel na odběratele (síťové zařízení `nfbXdY`), `count` je počet paketů, které chceme přijmout, `packets` je pole NDP paketů, tzn. segmentů obsahující NDP hlavičku a data. Funkce toto pole naplní až `count` novými pakety. První parametr `q` je přitom ukazatel na strukturu obsahující položky spojené s řízením přenosů pro daný DMA kanál, které tato funkce spravuje. Funkce umožňuje příjem paketů po dávkách (burst) podle parametru `count`, což zvýší výkon, především díky redukci s příjmem spojené režie (synchronizace ukazatelů do kruhového bufferu). Maximální výkon však v tomto případě hlavní priorita není a pakety se tak přijímají vždy po jednom. Po příjmu nových paketů se postupuje standardním způsobem. Nakonec je volána funkce `kndp_rx_burst_put()`, která provede synchronizaci ukazatele do kruhového bufferu a uvolní tak místo po paketu. Tento proces se neustále opakuje ve smyčce. V případě, že nejsou žádná nová data k dispozici, je vlákno uspáno na 1 ms, aby se ušetřil čas procesoru.

Odesílání paketů. Tato operace využívá funkci `kndp_tx_burst_get()`, která má totožné parametry s přijímací funkcí popsanou výše. Funkce ale pouze připraví místo v kruhovém bufferu. K jejímu volání je třeba vyplnit strukturu NDP paketu délkami NDP hlavičky a dat. Stejně jako u příjmu se vždy odesílá pouze jeden paket. NDP hlavička není v případě odesílání dat využita a její velikost je tak nulová. Po zavolání funkce `kndp_tx_burst_get` je ve struktuře NDP paketu vyplněn ukazatel na data. Přes něj jsou do kruhového bufferu zkopírována data ze struktury `sk_buff`. Podobně jako u ovladače nad rozhraním SZE2 je v případě paketů menších než konstanta `ETH_ZLEN` je zbylý prostor vyplněn nulami. Následuje funkce `ndp_tx_burst_flush()`, která sesynchronizuje ukazatele do kruhového bufferu a tím způsobí samotné odeslání paketů z karty. Nakonec se zaktualizují čítače paketů síťového rozhraní a uvolní se struktura `sk_buff`.

4.3 Aplikace pro hardwarový prefixový filtr

Čtení směrovacích i ARP cache tabulek patří mezi základní síťové úkony a v linuxových systémech pro to existují prověřené nástroje. Jedním z nich je nástroj `ip`, přesněji jeho varianta `ip monitor`, která umí monitorovat a vypisovat změny v směrovacích tabulkách i ARP cache. Zdrojové kódy jsou volně dostupné a rozhodl jsem se je použít jako základ pro vytvoření aplikace pro nahrávání pravidel do hardwarového prefixového filtru.

Program využívá knihovnu *libnetlink* pro práci s netlinkem, resp. rtnetlinkem, což je specializovaná část netlinku pro čtení a modifikaci směrovacích tabulek. Základem programu je otevřít rtnetlinkový socket funkcí `rtnl_open()` s deklarací, které typy zpráv se budou odebírat – v tomto případě pouze směrovací cesty a záznamy ARP cache. Poté se začne poslouchat funkcí `rtnl_listen()`. Při příchodu nové zprávy se zjistí její typ a zavolá se příslušná funkce. Ta obvykle zpracuje položky struktury, ve které je zpráva uchována a výsledek vypíše v čitelné formě. Následující výpis oznamuje, že ve směrovací tabulce 10 přibyl nový záznam směrovací cesty, kdy pakety s cílovým prefixem 99.99.99.0/24 mají být přeposlány na zařízení s IP adresou 147.229.177.156 přes síťové rozhraní nfb0d0. Druhý záznam říká, že zařízení s IP adresou 147.229.177.156 na síťovém rozhraní nfb0d0 je dostupné pod MAC adresou 00:25:90:93:7d:0e.

```
99.99.99.0/24 via 147.229.177.156 dev nfb0d0 table 10
147.229.177.156 dev nfb0d0 lladdr 00:25:90:93:7d:0e REACHABLE
```

Ze záznamů jsou k dispozici obě potřebné informace, tedy cílový prefix a MAC adresa next-hopu. Aplikace si vnitřně uchovává prefix, IP a MAC adresu next-hop směrovače v lineárním seznamu, který byl vybrán díky vhodným vlastnostem pro povahu aplikace (počet cest se může různě měnit, není důležité rychlé vyhledání prvku). V případě prvního záznamu se ve funkci zpracovávající tento typ zprávy (nová směrovací cesta) vytvoří nový prvek v seznamu a uloží se do něj informace o prefixu a IP adrese. V případě neznámé MAC adresy (nenachází se v ARP cache) se odešle ARP dotaz přes funkci `rtnl_talk()`. Při příchodu druhého záznamu se zkontroluje, zdali se v seznamu nachází záznam s IP adresou 147.229.177.156 a chybějící MAC adresou. Takový prvek se tam nachází a tak se mu vyplní chybějící MAC adresa. Pak už je možné pravidlo nahrát do firmwarového prefixového filtru přes knihovnu *dcprolib*. Sledování změn zajišťuje aktuálnost u nových a často měnících se záznamů, ale záznamy, které existovaly ve směrovací tabulce před spuštěním aplikace a nenastala u nich žádná změna, by však nebyly zaznamenány a proto také aplikace musí být schopna při

startu přečíst celou směrovací tabulku a poté také celou ARP cache, aby znala aktuální výchozí stav.

Kapitola 5

Testování a vyhodnocení

Implementace a testování ovladače probíhalo na vývojovém serveru Andre sdružení CESNET. Server Andre disponuje procesorem Intel Xeon E5-2609 s frekvencí 2,4 GHz, 16 GB operační pamětí a běží na linuxové distribuci Scientific Linux 7. Server je dále vybaven 100 gigabitovou COMBO kartou, která je fyzicky propojena se 100 gigabitovým rozhraním testovacího prostředí Spirent. V aplikaci Spirent TestCenter je možné vidět a analyzovat příchozí provoz z COMBO karty a zároveň aplikace umožňuje generovat až 100 gigabitový provoz do COMBO karty. Další vlastností aplikace je schopnost vytvářet virtuální zařízení s řadou podporovaných protokolů. Testovací prostředí je tak dobře připravené pro vyhodnocení síťových protokolů i měření výkonnosti ovladače.

5.1 Standardní síťové protokoly

Podkapitola se věnuje testování funkčnosti protokolů popsaných v podkapitole návrhu 3.1. U všech testů bylo nutné správně nastavit obecný hardwarový filtr, který musel provoz směrovaný přímo pro dané zařízení (DDoS Protector) vložit do správného DMA kanálu. Tedy takového kanálu, nad kterým bylo otevřeno síťové rozhraní.

5.1.1 ICMP, ARP, NDP

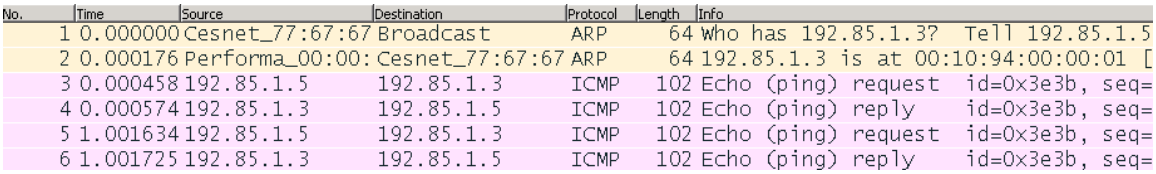
Pro otestování protokolu ICMP byl použit známý síťový diagnostický nástroj `ping`, který na cílové zařízení odesílá zprávy typu 8 – Echo Request a očekává odpověď typu 0 – Echo Request. V aplikaci Spirent TestCenter je proto vytvořeno virtuální síťové zařízení schopné odpovídat na ICMP zprávy. Je mu nastavena IP adresa 192.85.1.3 s maskou 255.255.255.0 (číslo prefixu 24) a MAC adresa 00:10:94:00:00:01. Vytvořenému síťovému rozhraní c0eth**0** je manuálně přiřazena IP adresa 192.85.1.5 se stejnou maskou 255.255.255.0, aby se zařízení nacházela ve stejné lokální síti. Jeho MAC adresa je 00:11:17:77:67:67. Obecný hardwarový filtr tedy byl nastaven tak, aby rámce s cílovou MAC adresou 00:11:17:77:67:67 přeposílal do DMA kanálu **0**. V příkazové řádce lze nastavení provést použitím příkazů:

```
ifconfig c0eth0 hw ether 00:11:17:77:67:67
ifconfig c0eth0 192.85.1.5 netmask 255.255.255.0
ifconfig c0eth0 up
```

Spuštění nástroje `ping` se provede příkazem `ping 192.85.1.3 -I c0eth0`, kde parametr -I označuje, z jakého síťového rozhraní se mají Echo Request dotazy odeslat. Před odesláním prvního dotazu se musí zjistit MAC adresa virtuálního zařízení, protože v ARP cache o ní

zatím neexistuje záznam. Jádro by mělo automaticky ARP dotaz provést a aktualizovat ARP cache. Lze tak zároveň otestovat funkčnost protokolu ARP.

Po spuštění nástroj `ping` opakovaně dostává odpověď a lze tak považovat protokoly ARP i ICMP za funkční. Na obrázku 5.1 je zachycena část této komunikace zobrazující počáteční ARP a ICMP komunikaci.

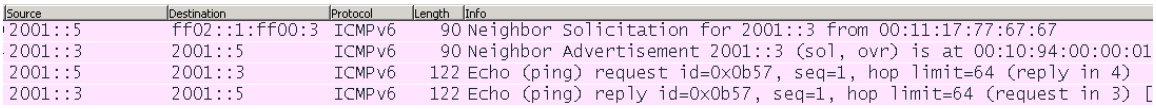


Obrázek 5.1: Záznam ICMP komunikace z programu Wireshark.

Otestování protokolu ICMPv6, jehož součástí je i protokol NDP, probíhalo podobným způsobem. Virtuálnímu zařízení v prostředí Spirent byla přiřazena IPv6 adresa 2001::3/64 a síťovému zařízení c0eth0 IPv6 adresa 2001::5/64. MAC adresy zůstaly zachovány.

```
sudo ifconfig c0eth0 inet6 add 2001::5/64
```

Stejně jako u ICMP pro verzi IPv4, po spuštění příkazu `ping -6 2001::3 -I c0eth0` se serveru Andre opakovaně dostává odpovědi a lze tak považovat protokol NDP (ICMPv6) za funkční. Na obrázku 5.2 je zachycena část této komunikace zobrazující počáteční zjištění MAC adresy virtuálního zařízení pomocí zpráv typu Neighbor Solicitation/Neighbor Advertisement a ICMPv6 požadavek a odpověď.



Obrázek 5.2: Záznam ICMPv6 komunikace z programu Wireshark.

Stejným způsobem byl otestován i ovladač pro variantu NFB ovladačů a bylo dosaženo stejných výsledků, tedy plné funkčnosti. Z důvodu stejného postupu i výsledků proto není dále rozepsán.

5.1.2 DHCP

Služba DHCP serveru není v konkrétním testovacím prostředí Spirent dostupná a tak se protokol DHCP testoval na vývojovém serveru Zinfandel. Ten je taktéž vybaven kartou COMBO a je fyzicky propojen s dalším serverem Pinot, který se bude chovat jako DHCP server. K vytvoření a spuštění DHCP serveru je použit program (démon) `dhcpd` nakonfigurovaný nad sítí 192.85.1.0/24 s dostupným adresním rozsahem 192.85.1.100-110. Na serveru Zinfandel se otevře síťové rozhraní c0eth0 bez předchozí konfigurace a proto mu bude chybět IP adresa. Chování DHCP klienta je zajištěno programem (démonem) `dhclient`, který je spuštěn nad síťovým rozhraním c0eth0. Program skončí po úspěšném získání IP adresy, v tomto případě 192.85.1.101, což lze ověřit příkazem `ifconfig` nebo `ip address`. Výpis z nástroje `ip address` je následující:

```
7: c0eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
           state UNKNOWN qlen 1000
```

```
link/ether 00:11:17:26:d8:38 brd ff:ff:ff:ff:ff:ff
inet 192.85.1.101/24 brd 192.85.1.255 scope global dynamic c0eth0
   valid_lft 473sec preferred_lft 473sec
```

Protokol DHCP je tedy funkční. Test nad ovladačem NFB ovladačů proveden nebyl z důvodu chybějícího firmware pro COMBO kartu zapojenou v režimu 10x10G. Dá se ale předpokládat, že pokud fungovaly předchozí protokoly (ICMP, ICMPv6, ARP), měl by fungovat i protokol DHCP, jelikož primární úlohou ovladače je pakety správně předat z/do síťové vrstvy jádra, což se v případě úspěšně otestovaných protokolů děje.

5.1.3 BGP, aplikace pro řízení prefixového filtru

Schopnost číst směrovací cesty protokolu BGP zajišťuje démon BIRD, který s nimi synchronizuje systémovou směrovací tabulku a tu čte aplikace pro řízení prefixového filtru `prfilterdaemon`. V první fázi testování byl vyzkoušen přepis směrovacích informací z BGP a démona BIRD do systémové směrovací tabulky. To proběhlo v rámci předpřipraveného prostředí v brněnském peeringovém uzlu BR-IX. Server, na kterém testy proběhly, ale neobsahoval COMBO kartu. Druhá fáze proběhla na serveru Andre, kde byl testován zápis pravidel do hardwarového prefixového filtru. Činnost příchozích BGP zpráv byla simulována manuálními zásahy do systémové směrovací tabulky. Aplikace nerozlišuje, kdo záznamy přidává či odebírá a proto mezi přidáním směrovací cesty démonem BIRD nebo manuální cestou není žádný rozdíl. Do prázdné směrovací tabulky 10 je přidána nová cesta, která pakety směřující do cílové sítě 99.99.99.0/24 přepošle na next-hop směrovač s IP adresou 192.85.1.2:

```
ip route add 99.99.99.0/24 via 192.85.1.2 table 10;
```

V aplikaci Spirent TestCenter je vytvořeno virtuální zařízení s IP adresou 192.85.1.2 a MAC adresou 00:10:94:00:11:01, které představuje směrovač. Dále je vytvořen generátor paketů, u kterého se nastaví cílová IP adresa na 99.99.99.99 a cílová MAC adresa na MAC adresu DDoS Protectoru, tedy 00:11:17:77:67:67 (viz podkapitola 5.1.1). Poté se nakonfiguruje a otevře síťové rozhraní c0eth0 (viz předchozí testování) a spustí se `prfilterdaemon` s parametry `-t 10 -i c0eth0`, kde parametr -t označuje směrovací tabulku s číslem 10 a parametr -i nastavuje síťového rozhraní, ze kterého se odešle ARP dotaz na next-hop směrovač. Pak je vygenerován a odeslán 1 paket. V aplikaci Spirent TestCenter je zachycena veškerá komunikace. Aplikace `prfilterdaemon` přečte cestu ze směrovací tabulky číslo 10 a odesílá ARP dotaz. Po získání odpovědi pravidlo nahraje do prefixového filtru. Karta COMBO příjme vygenerovaný paket, ten prochází skrze prefixový filtr a následně je odeslán na výstup do prostředí Spirent, avšak se změněnou cílovou MAC adresou 00:10:94:00:11:01, která odpovídá virtuálnímu next-hop směrovači. Při generování paketu s odlišnou cílovou IP adresou nebo smazáním příslušné cesty ve směrovací tabulce 10 se cílová MAC adresa změní podle výchozího pravidla. Dále bylo provedeno více testů s několika záznamy ve směrovací tabulce včetně IPv6, dynamického vkládání i odebírání za běhu. Aplikace `prfilterdaemon` je tak plně funkční. Test proběhl pouze nad ovladačem SZE2 rozhraní, u nových NFB ovladačů plná podpora prefixového filtru v knihovně *dcprolib* prozatím chybí.

5.2 Výkonnost ovladače

Karty COMBO jsou stále vyvíjeny. Pro potřeby vývoje a ladění existuje řada jednoduchých nástrojů. Některé z nich jsou v této práci také použity:

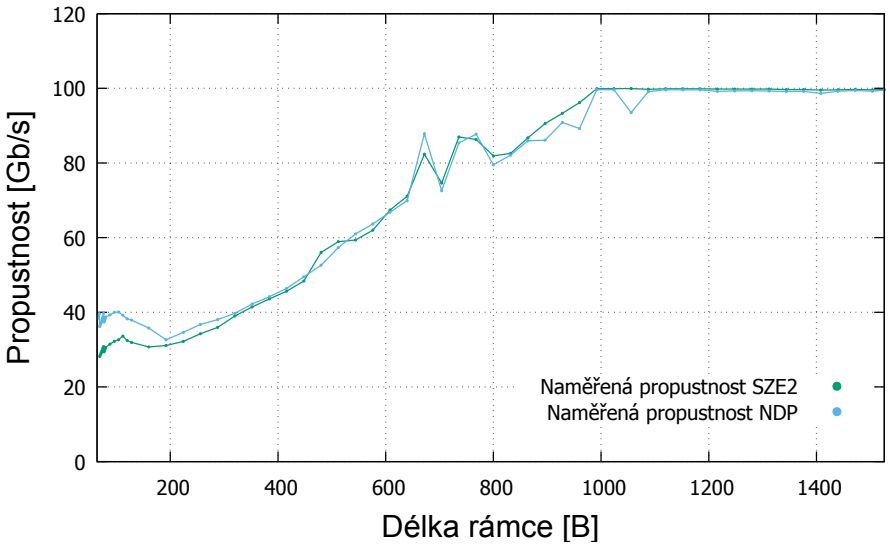
- `ibufctl` – zobrazuje informace o vstupním rozhraní (celkový počet bajtů, počet přijatých paketů, počet zahozených paketů...).
- `sze2multiread` – rychlé, vícevláknové čtení dat z RX DMA kanálů.
- `sze2fastwrite` – rychlé, vícevláknové odesílání paketů na TX DMA kanály.
- `nfb-eth` – nahrazuje `ibufctl` a `obufctl` u nových NFB ovladačů.
- `ndp-tool` – nahrazuje `sze2multiread` a `sze2fastwrite` u nových NFB ovladačů.
- `tcpreplay` – nepatří přímo k nástrojům pro karty COMBO, ale umožňuje rychle odesílat soubory ve formátu `*.pcap`.

V první části experimentálního měření výkonnosti jsou použity nástroje `sze2fastwrite` a `sze2multiread`, nebo `ndp-tool` pro vytvoření referenčních výsledků maximální dosažitelné propustnosti. Nástroje plně využívají možností SZE2/NDP rozhraní a obcházejí síťovou vrstvu, čímž dosahují vysoké propustnosti. V druhé části je provedeno měření průchodu dat přes ovladač a tedy i síťovou vrstvou jádra. Měření bylo provedeno ve směru příjmu (RX) i ve směru odesílání (TX). Měření ve směru RX spočívá ve čtení informací ze vstupního rozhraní o celkovém počtu příchozích paketů a počtu korektně přijatých paketů. Tyto informace lze vyčíst pomocí nástrojů `ibufctl` resp. `nfb-eth`. Pokud proces nestíhá číst data dostatečně rychle, kruhový buffer se celý zaplní. Tím dojde ke snížení počtu korektně přijatých paketů. Celková propustnost je poté určena na základě podílu počtu korektně přijatých a zahozených paketů. Měření dosažené propustnosti ve směru TX zajišťuje prostředí Spirent. Data jsou odesílána maximální rychlostí pomocí nástroje `sze2fastwrite`, `ndp-tool` resp. `tcpdump`. Dosažená maximální rychlost generování provozu je potom přímo zobrazena v prostředí aplikace Spirent TestCenter. K vyhodnocení výkonnosti a vykreslení grafů byly použity upravené skripty dostupné z interního repozitáře DDoS Protectoru. Skripty jsou napsané v jazyku TCL, využívají specializované API k ovládání prostředí Spirent a umožňují tak automatizované vyhodnocení propustnosti pro různé paketové délky. Pro vykreslení grafů byl následně použit nástroj `gnuplot`.

5.2.1 Směr příjmu (RX)

První měření na grafu 5.3 zachycuje propustnost dosaženou nástroji `sze2multiread` a `ndp-tool` při použití jediného RX DMA kanálu. Graf znázorňuje propustnost v Gb/s (osa y) v závislosti na délce rámce (osa x). Při velikosti rámce 1000 B a více oba nástroje dosahují plné propustnosti 100 Gb/s. Použitím více RX DMA kanálů by bylo možné dosáhnout 100 gigabitové propustnosti i při menších velikostech rámce. Měření je záměrně provedeno pouze pro jediný DMA kanál, neboť výsledná propustnost při použití více DMA kanálů odpovídá násobně pouze jejich počtu. V rámci zařízení DDoS Protector se přitom předpokládá, že z hlediska použití pouze pro nekritické operace/komunikaci bude vyhrazen pouze jediný DMA kanál, jehož propustnost bude plně dostačující.

V druhém měření na grafu 5.4, kde na ose x je délka rámce v bajtech a na ose y je propustnost v Gb/s, jsou vykreslené dvě varianty měření – modrá a zelená křivka zobrazuje průchod rámce síťovou vrstvou, zatímco oranžová a žlutá křivka zobrazuje upravenou verzi ovladače, v níž se rámec nepředá síťové vrstvě, ale všechny ostatní operace (alokace paměti, kopírování dat) jsou zachovány. Pokud nedojde k předání paketu síťové vrstvě, mají oba ovladače téměř identickou propustnost u rámců menších než přibližně 700 B. Od rámců větších jak 700 B začíná ovladač nad rozhraním SZE2 vykazovat mírně lepší propustnost než

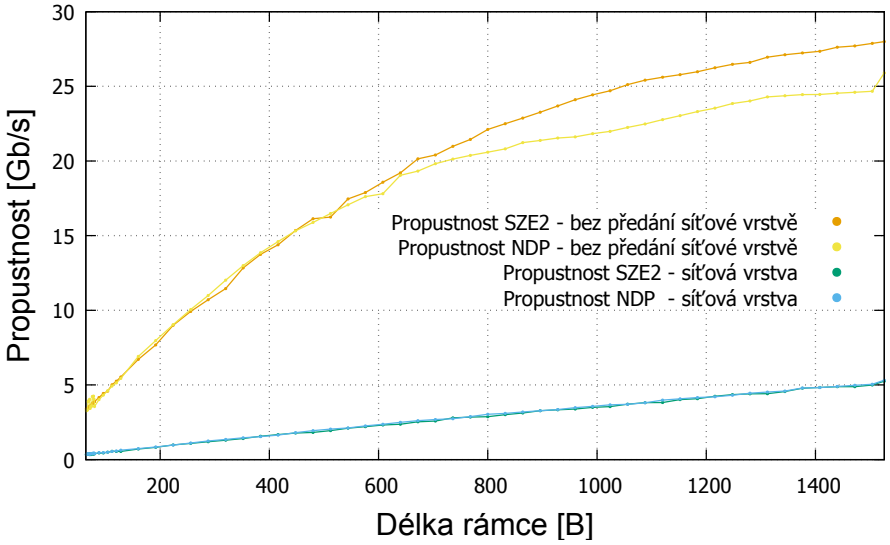


Obrázek 5.3: Referenční měření zachycující maximální možnou propustnost nad jediným RX DMA kanálem při použití nástrojů `sze2multiread` pro SZE2 a `ndp-tool` pro NDP rozhraní.

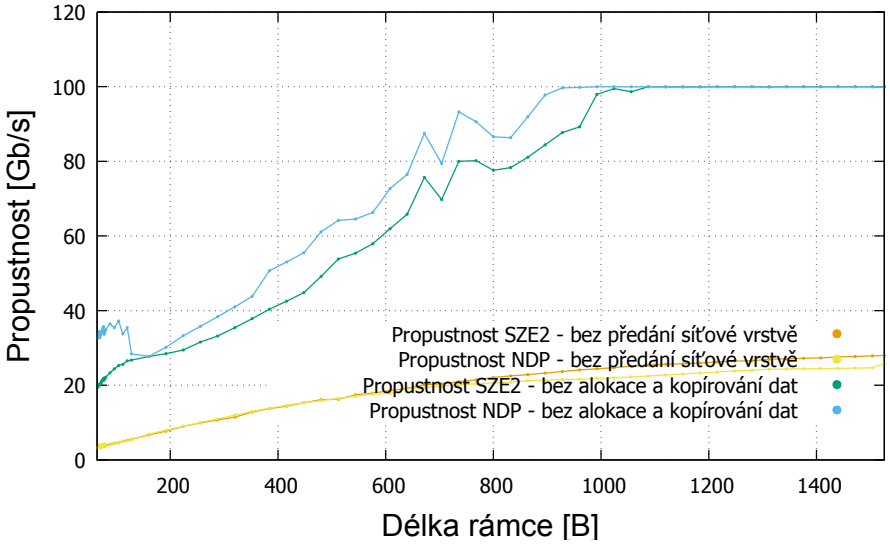
druhý ovladač a to až o 3 Gb/s. Pro rámce o velikosti 1500 B tak ovladač nad rozhraním SZE2 dosahuje propustnosti 28 Gb/s, zatímco ovladač nad rozhraním NDP propustnosti 25 Gb/s. Úzkým hrdlem při zpracování je však samotná síťová vrstva jádra, kde je pro rámec stejné velikosti dosaženo maximálně 5 Gb/s, což je zhruba o 80 % méně. Toto úzké hrdlo zároveň utlumuje mírně lepší výkon SZE2 varianty a oba ovladače tak mají identickou propustnost. Příjem dat by bylo možné dále optimalizovat, ale z hlediska zamýšlené aplikace pouze pro nekritickou část Protectoru je dosažený výkon plně dostačující. Na grafu 5.5 je modrou a zelenou barvou vyobrazena další, jinak upravená verze ovladačů, která odstraňuje i alokaci paměti, kopírování dat a obecně práci se strukturou `sk_buff`. Tato varianta se podobá referenčímu grafu 5.3. V případě NDP varianty tyto úpravy dosahují v jádru stejné výkonnosti jako přenos dat do uživatelské aplikace. Graf 5.5 tak potvrzuje, že pokles výkonu je dán právě limity jádra a síťové vrstvy, nikoliv samotného síťového rozhraní a jeho implementace. Naopak – při implementaci této části bylo dosaženo maximální možné propustnosti, kterou NDP rozhraní umožňuje. Plnohodnotný příjem rámců do jádra ale vyžaduje alokaci paměti pro každý rámec a kopírování paměti z kruhových bufferů namísto přímého mapování, jako je to možné u uživatelské aplikace, což vede k velmi výraznému poklesu výkonu.

5.2.2 Směr odesílání (TX)

První měření na grafu 5.6 zachycuje propustnost dosaženou nástroji `sze2fastwrite` resp. `ndp-tool` při použití jediného TX DMA kanálu. Jedná se tak o referenční měření zachycující maximální možnou propustnost. Graf znázorňuje propustnost v Gb/s (osa y) v závislosti na délce rámce (osa x). Při velikosti ethernetového rámce 400 B a více dosahují oba nástroje průměrně 90 Gb/s, zatímco pro nejkratší délku rámce 64 B se propustnost pro oba nástroje pohybuje kolem 40 Gb/s. Stejně jako u RX varianty by použitím více TX DMA kanálů bylo možné dosáhnout plné propustnosti i při menších velikostech rámce.

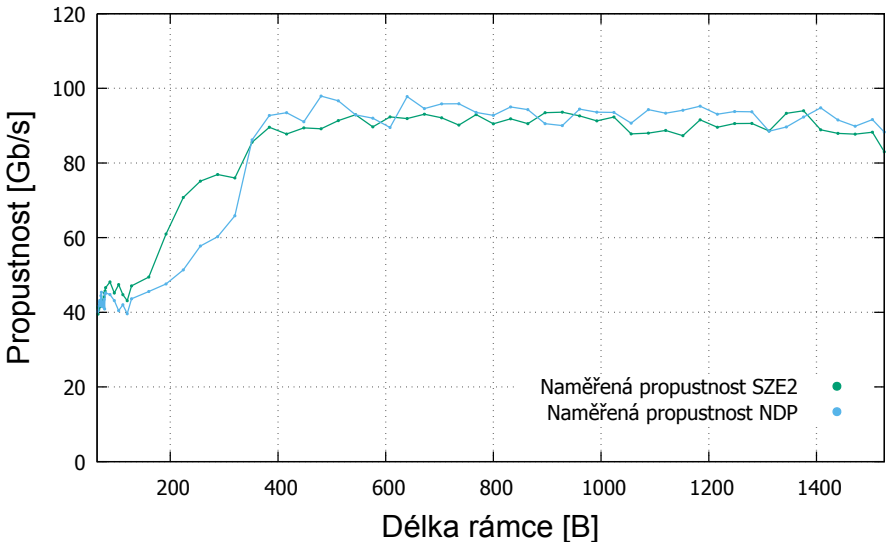


Obrázek 5.4: Srovnání dosažené propustnosti nad jediným RX DMA kanálem mezi ovladači pro SZE2 a NDP rozhraní. Graf zároveň zachycuje vliv zpracování rámce síťovou vrstvou na celkovou propustnost.



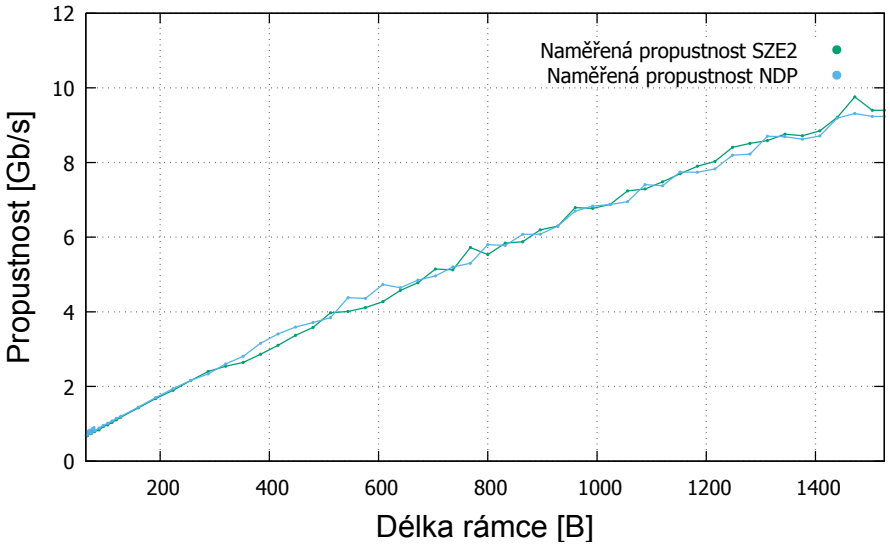
Obrázek 5.5: Srovnání dosažené propustnosti nad jediným RX DMA kanálem mezi modifikovanými ovladači pro SZE2 a NDP rozhraní. Modifikace odstraňuje alokaci paměti a kopírování dat při každém rámci, a díky tomu dosahuje téměř referenčních výsledků. Graf také obsahuje variantu ovladače bez předání rámce síťové vrstvě z předchozího grafu 5.4 pro porovnání.

Druhé měření na grafu 5.7, kde na ose x je opět délka rámce v bajtech a na ose y je propustnost v Gb/s, využívá nástroj `tcpreplay`, kterému se předá soubor ve formátu `*.pcap` obsahující rámce požadované velikosti. Ten poté co nejrychleji rámce ze souboru odesílá přes síťové rozhraní nfb0d0/c0eth0 vytvořené implementovaným ovladačem. Jak je vidět z grafu 5.7, s rostoucí délkou rámce se téměř lineárně zvyšuje i výkon, ale propustnost



Obrázek 5.6: Referenční měření zachycující maximální možnou propustnost nad jediným TX DMA kanálem při použití nástrojů `sze2fastwrite` pro SZE2 a `ndp-tool` pro NDP rozhraní.

je oproti referenčnímu měření výrazně nižší. Při maximální velikosti ethernetového rámce 1526 B je dosaženo pouze kolem 9 Gb/s. To je zhruba 10x méně, než v případě referenčního měření a pro nejkratší rámce o velikosti 64 B je propad výkonu ještě výraznější, dosahující až 40x nižší propustnosti. Propad je způsoben tím, že i v případě odesílání dat prostřednictvím ovladače dochází ke kopírování dat, alokaci a dealokaci paměti, zatímco referenční měření využívá mapování paměti přímo do adresového prostoru uživatelské aplikace, takže k nutnosti kopírování dat dochází minimálně, jelikož jsou jednotlivé rámce zapisovány přímo do kruhového bufferu. U nástroje `tcpreplay` byl mj. použit parametr `-K`, který soubory před zpracováním nahraje do paměti RAM, aby bylo dosaženo maximálního výkonu při generování dat.



Obrázek 5.7: Srovnání dosažené propustnosti nad jediným TX DMA kanálem mezi ovladači pro SZE2 a NDP rozhraní.

Kapitola 6

Závěr

Cílem práce bylo provést návrh a implementaci ovladače síťových karet rodiny COMBO umožňujícího příjem a odesílání paketů prostřednictvím standardního síťového rozhraní jádra Linux. Při řešení práce jsem se v rámci teoretického rozboru problému seznámil s FPGA síťovými kartami rodiny COMBO a s aktuálním programovým rozhraním SZE2 pro příjem ethernetových rámců ze sítě. Rozhraní SZE2 využívá při příjmu paketů ze sítě přenosů DMA a přímého mapování paměti kruhových bufferů do paměťového prostoru uživatelské aplikace, čímž zcela obchází síťovou vrstvu v jádře. Nastudoval jsem informace o zařízení vyvíjeném v rámci sdružení CESNET pro ochranu před DDoS útoky, k jehož rozšíření budou využity výsledky této práce. Dále jsem nastudoval obecné principy síťových protokolů a zaměřil se především na protokoly ARP, NDP, ICMP, DHCP a částečně i BGP, na jejichž podporu se cílí v případě zařízení Protector při nasazení do L2 infrastruktury. Nakonec jsem se seznámil s problematikou vývoje ovladačů pracujících v režimu jádra a z toho vyplývajících odlišností oproti vývoji uživatelských aplikací.

Po zpracování teoretického základu jsem porovnával dva možné přístupy pro zajištění příjmu síťových dat nad rozhraním SZE2, zdůvodnil výběr varianty ovladače síťového zařízení a provedl návrh takového ovladače, zahrnující standardní operace jako otevření síťového rozhraní a příjem/odesílání dat. Jelikož v průběhu řešení bakalářské práce byla zveřejněna nová generace rozhraní pro COMBO karty, rozhraní NDP, provedl jsem nad rámec zadání i návrh ovladače pro rozhraní NDP. Oba ovladače jsem následně podle návrhu implementoval. S využitím vytvořeného ovladače jsem také navrhl aplikaci pro příjem směrovacích informací přes protokol BGP a jejich synchronizaci s firmwarovou komponentou pro akcelerované přeposílání paketů. K příjmu směrovacích informací je přitom využit open-source směrovací démon BIRD, který přijaté informace synchronizuje se systémovými směrovacími tabulkami, které je možné jednoduše číst.

Oba vytvořené ovladače i aplikaci pro řízení akcelerovaného přeposílání paketů jsem otestoval. Ověřil jsem funkčnost pro jednotlivé protokoly ARP, NDP, ICMP a DHCP. Následně jsem provedl měření dosažené propustnosti. Oba ovladače dosahovaly téměř stejných výsledků. Ve směru příjmu byla naměřena propustnost 5 Gb/s při velikosti rámce 1526 B a ve směru odesílání 9 Gb/s při stejné velikosti rámce. Dosažená propustnost je výrazně nižší než maximální možná propustnost (100 Gb/s), kterou umožňuje samotné rozhraní SZE2 resp. NDP. Dalším měřením bylo potvrzeno, že úzkým hrdlem je provádění alokace paměti a kopírování dat pro každý paket, což je ale nutnou součástí přenosu dat do síťové vrstvy jádra. Z hlediska využití ovladače pouze pro nekritické operace DDoS Protectoru je však dosažená propustnost plně dostačující.

Výsledky této práce mají praktické uplatnění, kdy ovladač i aplikace pro řízení akcelerovaného přeposílání paketů budou přímo využity u zařízení DDoS Protector. V rámci pokračování práce bude probíhat integrace ovladače i aplikace do infrastruktury a dalšího obslužného software DDoS Protectoru. Dále se očekává pilotní nasazení nové verze DDoS Protectoru v páteřní síti CESNET. Z hlediska dalšího rozvoje ovladače je možné se blíže zaměřit na optimalizaci výkonnosti, bude-li to potřeba, případně se zaobírat možností složitější konfigurace mapování síťové rozhraní-DMA kanál-fyzické síťové rozhraní. V této práci byla totiž implementována pouze varianta síťového rozhraní pro každý pár RX/TX DMA kanálů, ale pro některé typy aplikací může být výhodnější použít i jiný typ mapování.

Literatura

[1] BIRD: *BGP*. [Online; navštíveno 09.04.2018].
URL http://bird.network.cz/?get_doc&v=20&f=bird-6.html#ss6.3

[2] BIRD: *Introduction*. [Online; navštíveno 09.04.2018].
URL http://bird.network.cz/?get_doc&v=16&f=bird-1.html#ss1.1

[3] BIRD: *The BIRD Internet Routing Daemon*. [Online; navštíveno 09.04.2018].
URL http://bird.network.cz/

[4] Droms, R.: Dynamic Host Configuration Protocol. RFC 2131, RFC Editor.
URL https://www.ietf.org/rfc/rfc2131.txt

[5] ExamCollection: *Comparing OSI and TCP/IP models*. [Online; navštíveno 09.04.2018].
URL https://www.examcollection.com/certification-training/network-plus-comparing-osi-and-tcp-model.html

[6] Štěpán FRIEDL; Puš, V.; Matoušek, J.; aj.: *Designing a Card for 100 Gb/s Network Monitoring*. Technická zpráva, CESNET, 2013, [Online; navštíveno 25.01.2018].
URL https://www.cesnet.cz/wp-content/uploads/2014/02/card.pdf

[7] Hank, A.: *Návrh síťových aplikací na platformě Netcope*. Diplomová práce, Vysoké učení technické v Brně, 2009, [Online; navštíveno 25.01.2018].
URL http://www.fit.vutbr.cz/study/DP/DP.php.cs?id=8195&file=t

[8] IEEE: OUI. [Online; navštíveno 13.05.2018].
URL http://standards-oui.ieee.org/oui.txt

[9] Jelínek, L.: *Jádro systému Linux: Kompletní průvodce programátora*. Computer Press, 2008, ISBN 978-80-251-2084-2.

[10] Kuka, M.: *Hardwarově akcelerované zařízení pro ochranu před DoS útoky*. Diplomová práce, Vysoké učení technické v Brně, 2017, [Online; navštíveno 18.04.2018].
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=19924&file=t

[11] Kučera, J.; Šiška, P.; Kuka, M.; aj.: *DDoS Protector User Manual*. LIBEROUTER / CESNET TMC group, 2017.

[12] Liberouter: *Cards*. [Online; navštíveno 25.01.2018].
URL https://www.liberouter.org/technologies/cards/

[13] Liberouter: *Cards gallery*. [Online; navštíveno 25.01.2018].
URL https://www.liberouter.org/cards-gallery/

[14] Matoušek, P.: *Síťové aplikace a jejich architektura*. Vutium, 2014, ISBN 978-80-2143-766-1.

[15] Narten, T.; Nordmark, E.; Simpson, W. A.; aj.: Neighbor Discovery for IP version 6 (IPv6). RFC 4861, RFC Editor.
URL https://tools.ietf.org/html/rfc4861

[16] NIX.CZ: NIX.CZ – Neutral Internet eXchange. [Online; navštíveno 09.05.2018].
URL https://www.nix.cz/cs

[17] Paluch, P.: ARP packet size. [Online; navštíveno 05.05.2018].
URL https://supportforums.cisco.com/t5/lan-switching-and-routing/arp-packet-size/td-p/1551467

[18] Plummer, D. C.: An Ethernet Address Resolution Protocol. RFC 826, RFC Editor.
URL https://tools.ietf.org/html/rfc826

[19] Postel, J.: INTERNET CONTROL MESSAGE PROTOCOL. RFC 792, RFC Editor.
URL https://tools.ietf.org/html/rfc792

[20] Rekhter, Y.; Li, T.; Hares, S.: A Border Gateway Protocol 4 (BGP-4). RFC 4271, RFC Editor.
URL https://tools.ietf.org/html/rfc4271

[21] Ráb, J.; Ryšavý, O.; Veselý, V.: LAN, Ethernet. [Online; navštíveno 09.04.2018].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIPK-IT%2Flectures%2F2016%2Fipk2017L-p08-linkova.pdf&cid=11465

[22] Vido, M.: *DPDK nad síťovými kartami COMBO*. Diplomová práce, Vysoké učení technické v Brně, 2016, [Online; navštíveno 25.01.2018].
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=18312&file=t

[23] Wikipedia contributors: OSI model — Wikipedia, The Free Encyclopedia. [Online; navštíveno 09.04.2018].
URL https://en.wikipedia.org/wiki/OSI_model

[24] Wikipedia contributors: PCI Express — Wikipedia, The Free Encyclopedia. [Online; navštíveno 09.04.2018].
URL https://en.wikipedia.org/wiki/PCI_Express

[25] Wikipedia contributors: Programovatelné hradlové pole — Wikipedia, The Free Encyclopedia. [Online; navštíveno 09.04.2018].
URL
https://cs.wikipedia.org/wiki/Programovateln%C3%A9_hradlov%C3%A9_pole

[26] Xilinx: *FPGAs & 3D ICs*. [Online; navštíveno 17.04.2018].
URL http://www.xilinx.com/products/silicon-devices/fpga.html

[27] Žádník, M.: *DDoS Protector aneb čistička*. [Online; navštíveno 06.04.2018].
URL https://www.cesnet.cz/wp-content/uploads/2017/02/ddos-protector.pdf