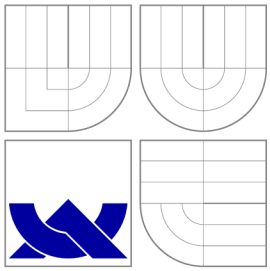
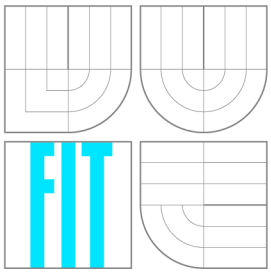


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

STAVOVÉ ZPRACOVÁNÍ TCP/IP TOKŮ

STATEFULL PROCESSING OF TCP/IP FLOWS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN KOŠEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KOŘENEK
SUPERVISOR

BRNO 2007

Zadání bakalářské práce

1. Seznamte se s kartou COMBO6X a technologií Virtex-II Pro od firmy Xilinx.
2. Nastudujte síťové protokoly TCP/IP a problematiku skládání TCP/IP toků. Navrhněte architekturu komponent, která umožní stavové zpracování TCP/IP toků. Při návrhu se snažte o dosažení maximální flexibility mezi velikostí uchovávaných informací a počtem zpracovávaných toků.
3. Proveďte implementaci navržené architektury v jazyce VHDL s ohledem na syntézu do FPGA.
4. Nad vytvořenou implementací proveďte syntézu a zjistěte velikost a maximální frekvenci výsledného obvodu.
5. Funkci navrženého řešení ověřte nad aplikací hledání řetězců v TCP/IP tocích. Jako cílovou platformu použijte kartu COMBO6X.
6. V závěru diskutujte dosažené výsledky a uveďte možné uplatnění navrženého řešení.

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Bezpečnostní síťová zařízení se stávají nezbytnou součástí mnoha univerzitních nebo komerčních sítí. Pro dosažení potřebné úrovně bezpečnosti však zmíněná zařízení pro svoji činnost vyžadují technologie komplexní analýzy provozu, jako je stavové filtrování nebo rekonstrukce TCP toků. Tato bakalářská práce se zabývá návrhem a implementací flexibilní síťové platformy pro stavové zpracování toků. Umožňuje analyzovat a zpracovávat vstupní data přímo na úrovni datových toků, nejen nad pakety. Navržená architektura je díky své flexibilitě vhodná pro široké spektrum aplikací, zajišťuje rozdělování výkonu a konzistentní zpracování stavové informace. Výhody tohoto přístupu jsou demonstrovány na několika síťových aplikacích.

Klíčová slova

stavové zpracování toků, bezpečnost sítí, Internet, FPGA, VHDL

Abstract

Network security systems become an essential part of many network structures in both company and university domains. These systems however require a higher semantic level of network traffic analysis like statefull filtration or TCP stream reassembling. This bachelor work deals with an architecture of flexible network platform capable of statefull processing at multigigabit speeds. It allows to analyze and process incoming network traffic with a flow-based approach rather than packet-based one. The proposed architecture is flexible in supporting wide range of applications, allows performance scalability and state information consistency checking. The advantages and flexibility of proposed platform is demonstrated on several network security applications.

Keywords

statefull flow processing, network security, Internet, FPGA, VHDL

Citace

Martin Košek: Stavové zpracování TCP/IP toků, bakalářská práce, Brno, FIT VUT v Brně, 2007

Stavové zpracování TCP/IP toků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Kořenka. Další informace mi poskytli kolegové z projektu Liberouter. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Košek
12. května 2007

Poděkování

Především bych chtěl poděkovat vedoucímu své bakalářské práce panu Ing. Janu Kořenkovi za odborné vedení a čas věnovaný konzultacím této práce. Také bych chtěl poděkovat kolegům z projektu Liberouter za poskytnutí informací a zajištění technické podpory při návrhu a implementaci.

Obsah

Kapitola 1

Úvod

V poslední době jsme svědky výrazného rozvoje Internetu. Počet uživatelů a poskytovaných služeb rychle roste a s ním roste i potřeba analyzovat a zpracovávat přenesené informace. Vzhledem ke zvyšující se četnosti útoků a jejich stále obtížnějšímu odhalení se klade důraz na vývoj zařízení schopných komplexní analýzy a zpracování síťového provozu. Při vývoji těchto zařízení nelze využít klasické zpracování založené na programovém vybavení, protože taková zařízení by nebyla schopna zpracovat data na multigigabitových rychlostech (1 až 10 Gb/s). Tato zařízení jsou limitována výkonem procesoru a propustností systémové sběrnice. Zpracování je nutné přesunout na nižší vrstvu – hardware, která poskytne potřebný paralelismus a dostatečný výpočetní výkon. Síťová zařízení mohou být s výhodou vyvinuta na programovatelných hradlových polích (FPGA), která jsou flexibilní a poskytují dostatečný výpočetní výkon pro zpracování nebo analýzu provozu na multigigabitových sítích.

Pro mnoho bezpečnostních síťových zařízení je nezbytné *stavové* zpracování příchozího provozu, kdy je analýza prováděna nad celým tokem, ne pouze nad jednotlivými pakety [10]. Jedná se například o aplikace typu vyhledávání řetězců, systémy pro analýzu protokolů, datových toků a dalších bezpečnostních systémů, kde je potřeba pracovat se stavovou informací přiřazenou k danému datovému toku. Pro demonstraci důležitosti stavového zpracování poslouží systém detekce nežádoucího provozu IDS – Intrusion Detection System. Zařízení má za úkol analyzovat síťovou komunikaci a vyhledat v datech paketu vzory, které mohou indikovat útok. Bez možnosti ukládat aktuální stav vyhledávajícího automatu ale tento systém není schopen nalézt řetězce rozdělené mezi více paketů. Tato slabina může být zneužita potenciálním útočníkem umístěním detekovatelného řetězce mezi více paketů, čímž prolomí ochranu detekčního systému.

Z těchto poznatků vyplývá, že při vývoji systému pro stavové zpracování toků (dále jen *platformy*) je třeba brát ohled na cílovou aplikaci, která může mít různé požadavky. Některé aplikace mohou vyžadovat ukládání rozsáhlé stavové informace, ale nevyžadují vysokou rychlost zpracování, zatímco pro jiné aplikace může být prioritou analýza provozu na vysokých rychlostech (až 10 Gb/s). Pro dostatečnou flexibilitu by měla platforma umožňit zvolení libovolné externí paměti pro cílovou aplikaci.

Velikost kontextu by taktéž měla být volitelná, protože se opět jedná a vlastnost specifickou pro každou aplikaci. Například zařízení typu IDS bude zpravidla vyžadovat kontext malé velikosti pouze na uložení stavu vyhledávacího automatu, zatímco systémy pro analýzu datových toků mohou ukládat celé bloky dat. Další velmi důležitou vlastností každé aplikace je počet procesních jednotek. Zpracování paketu může vyžadovat větší množství výpočetního času a jedna procesní jednotka nemusí mít dostatečnou propustnost pro zpracování vstupního síťového toku. Z tohoto důvodu by měla platforma podporovat zcela voli-

telný počet procesních jednotek. S větším počtem procesních jednotek ale dochází k několika problémům. Například by mohlo dojít k použití nekonzistentní stavové informace, které by mohlo vést k omezení, či úplnému znemožnění zpracování dat. Tento problém je dále vysvětlen v kapitole 4.4.1 a je ukázáno jeho řešení.

Dokument je logicky členěn do několika kapitol. V teoretické části je čtenář seznámen s referenčním síťovým modelem ISO/OSI, který je dále odkazován při vysvětlení způsobu identifikace toků. V kapitole 3 je proveden rozbor současného stavu a předvedeno několik současných přístupů ke stavového zpracování. U jednotlivých přístupů jsou vysvětleny jejich výhody a nevýhody a nakonec jsou tyto systémy srovnány s vyvíjenou platformou. Kapitola 4 se zabývá vlastním návrhem a implementací architektury platformy. Jsou zde uvedeny blokové diagramy jednotlivých komponent a vysvětlena jejich funkce. V kapitole 5 jsou předvedeny výsledky získané po implementaci platformy a analýz důležitých vlastností aplikace jako je propustnost platformy při použití různých typů pamětí nebo využití zdrojů na FPGA. Výhody vyvíjené platformy jsou předvedeny na několika síťových aplikacích v kapitole 6. V závěru jsou zhodnoceny dosažené cíle a diskutovány možnosti dalšího pokračování práce.

Kapitola 2

Teoretický úvod

2.1 Referenční síťový model ISO/OSI

Referenční síťový model ISO/OSI byl vytvořen mezinárodní organizací ISO (International Organization for Standardization) jako jednotný standard pro vzájemné propojování různých systémů. Standard definuje celkem sedm hierarchických vrstev, které jsou znázorněny na obrázku 2.1.

Obrázek 2.1: Referenční model ISO/OSI

Ve standardu je definováno přesné rozhraní služeb, které jednotlivé vrstvy poskytují. Vzhledem k jeho komplikovanosti se ale dnes většinou využívá pouze jako reference. V následujících bodech jsou popsány funkce jednotlivých vrstev.

Fyzická vrstva se zabývá přenosem bitů komunikačním kanálem, ale nevěnuje už pozornost významu přenášených bitů. Specifikuje přenosové médium, úrovně napětí hodnoty logické jedničky nebo nuly, délku doby pro přenos jednoho bitu a další parametry, které určují mechanické a elektrické rozhraní. Příkladem rozhraní je RS232.

Linková vrstva vytváří nad fyzickou vrstvou datový spoj. Standardně je datový spoj vytvářen mezi bezprostředními sousedy, tedy na dvoubodovém spoji. Linková vrstva organizuje bitový proud fyzické vrstvy do rámců, což jsou obvykle bloky dat o velikosti desítek až tisíců bajtů. Rámec má svou přesnou strukturu, je v něm obvykle obsažena

zdrojová a cílová adresa rámce. Vrstva umožňuje rozpoznávat chybně přenesené rámce a zajistit jejich opravný přenos.

Síťová vrstva řeší směrování toku dat v síti od odesílatele k adresátovi. Zdrojový a cílový uzel přitom nemusí být přímo propojeny. Hlavním úkolem této vrstvy je řešit problematiku směrování. Pro tyto účely mohou být data děleny na pakety. Pro zajištění toku dat sítí je potřeba jednoznačná identifikace uzlu, která je nezávislá na fyzické adresaci. Tato identifikace se nazývá síťová adresa a je přiřazována všem uzlům sítě. Služby síťové vrstvy se dělí na spojované, které jsou spolehlivé, a na nespojované, které spolehlivost doručení dat nezaručují. Součástí této vrstvy je například IP protokol.

Transportní vrstva zaručuje adresování koncových komunikujících prvků, které působí v jednotlivých uzlech sítě. Mezi tyto prvky patří například procesy nebo uživatelské relace. Data jsou přenášena po blocích (segmentech). V segmentu je adresa, která umožňuje identifikaci komunikujících prvků v rámci jednotlivého uzlu. Na úrovni síťové vrstvy jsou segmenty rozděleny na pakety, do kterých se přidají adresy komunikujících uzlů. Po uskutečnění přenosu jsou segmenty z paketů opět sestaveny. Na úrovni transportní vrstvy mohou existovat spojované i nespojované služby. U spojovaných služeb je v této vrstvě zajištěno ustavení komunikace a spolehlivé doručení dat. Naopak u nespojovaných služeb se o spolehlivé doručení dat musí postarat sama aplikace. Příkladem protokolu této vrstvy je TCP (spojovaná služba) nebo UDP (nespojovaná služba).

Relační vrstva umožňuje procesům nebo koncovým uživatelům v různých uzlech sítě ustanovení relací, pomocí kterých pak koordinují svoji činnost.

Prezentační vrstva obsahuje funkce, které jsou prováděny tak často, že je pro ně vhodné mít obecné řešení. Uživatelé pak nemusí tyto funkce řešit ve vlastní režii. Mezi funkce prezentační vrstvy patří typicky převody mezi různými kódováními, komprimace nebo šifrování dat.

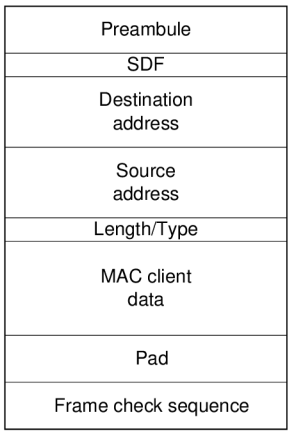
Aplikační vrstva realizuje aplikačně orientované služby. Poskytuje tedy různá aplikační rozhraní jako například služby pro implementaci elektronické pošty, přenosu souborů nebo elektronického obchodu. Součásti této vrstvy bývají přímo procesy, které tyto aplikační funkce plní.

2.2 Linková vrstva a Ethernet

Jedním z nejrozšířenějších protokolů, který pracuje na linkové vrstvě, je Ethernet. Protokol zajišťuje komunikaci mezi dvěma sousedními uzly sítě a řeší přenos dat na úrovni jednoho spoje. Na linkové vrstvě jsou řešeny i metody přístupu ke společnému médiu. Ethernet používá metodu CSMA/CD [7].

Při komunikaci jsou data vyšších vrstev vkládána do bloků dat, které se nazývají *rámce* (frames). Začátek a konec každého rámce je při přenosu signalizován speciální značkou. Formát rámce je znázorněn na obrázku 2.2.

Preambule (7 bajtů) – vzorek dat, který slouží k zajištění časové synchronizace při příchodu rámce.



Obrázek 2.2: Struktura ethernetového rámce

Start Frame Delimiter (SDF) (1 bajt) – identifikuje začátek rámce.

Destination address (6 bajtů) – adresa cílového uzlu, pro který je rámec určen. Může se jednat o adresu konkrétního uzlu nebo o multicast. V takovém případě je rámec určen pro více uzlů.

Source address (6 bajtů) – adresa zdrojového uzlu.

Length/Type (2 bajty) – význam položky se mění s hodnotou, která je v ní uložena. Pokud je hodnota větší jak 1536 (600h), obsahuje typ dat vyšší vrstvy (MAC Client Data). Pokud je hodnota menší, tak je v položce uložena velikost dat.

MAC data a Pad (46 až 1500 bajtů) – zde jsou uložena data protokolů vyšších vrstev. Pokud je velikost dat příliš malá, je oblast rozšířena o položku Pad, která může obsahovat libovolná data. Tato položka zajišťuje minimální délku rámce, která je nutná pro správnou funkci metody CSMA/CD.

Frame check sequence (4 bajty) – obsahuje kontrolní součet rámce, který se počítá nad celým rámcem kromě položek *Preambule*, *SDF* a samozřejmě kontrolního součtu.

2.3 Síťová vrstva a Internet Protocol

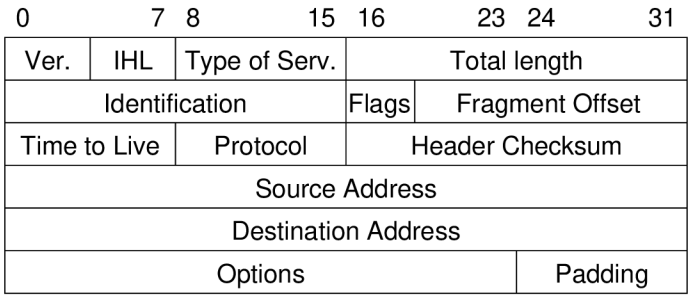
IP protokol je součástí síťové vrstvy referenčního modelu ISO/OSI. Jedná se o datagramový protokol zajišťující přenos dat ze zdrojového do cílového uzlu. Jeho specifikaci je možné najít v RFC 791 [12].

Dnes je běžně používán IP protokol verze 4 (IPv4). U tohoto protokolu má každý uzel svou jedinečnou IP adresu, která je strukturována podle tříd adres na identifikaci sítě a identifikaci uzlu v síti. Formát IPv4 hlavičky je uveden na obrázku 2.3.

Version (4 bity) – definuje verzi protokolu. Pro IPv4 je tato položka rovna hodnotě 4.

Internet Header Length (4 bity) – počet 32-bitových slov v IPv4 hlavičce.

Type of Service (8 bitů) – typ služby je položka, která informuje o požadované kvalitě služeb při přenosu (QoS).



Obrázek 2.3: Formát IPv4 hlavičky

Total Length (16 bitů) – celková délka datagramu v bajtech (včetně hlavičky).

Identification (16 bitů) – identifikační hodnota, kterou vkládá odesílatel při fragmentaci datagramu. Tato hodnota se využije při zpětném sestavování datagramu.

Flags (3 bity) – příznaky určující možnost fragmentace datagramu a její stav. Jedná se o příznaky *Don't Fragment* (zákaz fragmentace) a *More Fragments* (další fragmenty).

Fragment Offset (13 bitů) – určuje pozici fragmentu v rámci datagramu. Pozice se udává v násobcích 8 bajtů.

Time to Live (8 bitů) – původně měla tato položka význam životnosti paketu v sekundách. V současnosti je významem TTL počet aktivních prvků v síti, přes které může datagram projít. Každý aktivní prvek, kterým datagram projde, tuto hodnotu sníží o jedna. Pokud TTL dosáhne nuly, datagram je zahozen.

Protocol (8 bitů) – identifikuje protokol vyšší vrstvy, který je přenášen v datech datagramu.

Header Checksum (16 bitů) – kontrolní součet IPv4 hlavičky.

Source Address (32 bitů) – adresa zdrojového uzlu.

Destination Address (32 bitů) – adresa cílového uzlu.

Options – volitelné položky, které se mohou nepovinně vyskytovat v IPv4 hlavičce. Délku tohoto pole může zpracovávající uzel zjistit z položky IHL.

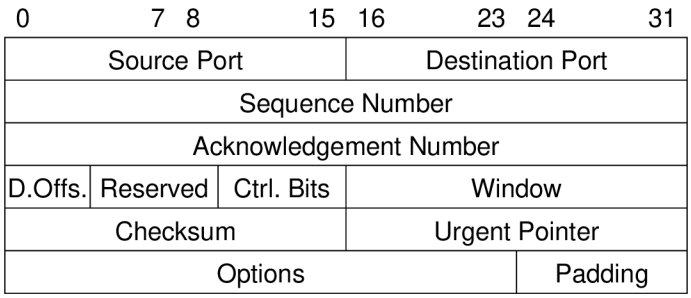
Padding – pokud není délka IPv4 hlavičky dělitelná 32 bity, obsahuje tato položka zarovnání na tuto délku. Zarovnání se skládá z nulových bitů.

U protokolu IPv4 se pro adresování uzlu v síti používá jedinečná adresa. Šířka této adresy je 32 bitů, což v současné době neposkytuje dostatečný adresový prostor. Pro odstranění tohoto problému byl vytvořen protokol IPv6. Adresový prostor byl rozšířen z původních 32 na 128 bitů a vylepšen z pohledu počtu a organizace úrovní adresové hierarchie. Podrobný popis tohoto protokolu není z důvodu zaměření této práce nutný, zájemci si mohou specifikaci protokolu vyhledat v RFC 2460 [5].

2.4 Transportní vrstva

2.4.1 Transmission Control Protocol

TCP protokol [13] přijímá data z relační vrstvy jako proud dat, seskupuje do číslovaných paketů a odesílá ve správném pořadí k cílovému uzlu. Jedná se o stavový spojovaný protokol, který zajišťuje spolehlivé doručení paketů ve správném pořadí. Rovněž se stará o řízení toku jako obranu proti zahlcení cílového uzlu. Formát TCP hlavičky je uveden na obrázku 2.4.



Obrázek 2.4: Formát TCP hlavičky

Source Port (16 bitů) – číslo zdrojového portu.

Destination Port (16 bitů) – číslo cílového portu.

Sequence Number (32 bitů) – udává pořadové číslo prvního datového bajtu v tomto segmentu.

Acknowledgement Number (16 bitů) – pokud je nastaven příznak ACK, obsahuje tato položka následující pořadové číslo, které je příjemce připraven přijmout.

Data Offset (4 bity) – udává počet 32-bitových slov v TCP hlavičce.

Reserved (6 bitů) – rezervováno pro možná budoucí použití. Musí vždy obsahovat nulovou hodnotu.

Control Bits (6 bitů) – pole příznaků. Z hlediska této práce jsou důležité pouze příznaky ACK (platná položka Acknowledgement), SYN (odesílatel začíná novou sekvenci číslování) a FIN (ukončení přenosu v jednom směru – odesílatel již nepošle žádná další data).

Checksum (16 bitů) – kontrolní součet vypočítaný z některých položek IP hlavičky a celého TCP segmentu.

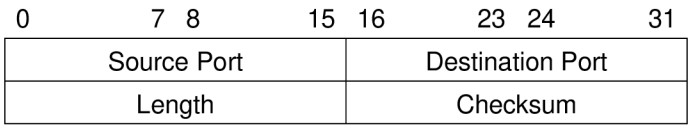
Urgent Pointer (16 bitů) – ukazatel na konec úseku naléhavých dat. Tato položka se bere v potaz pouze pokud je nastaven odpovídající příznak (URG).

Options – volitelné položky, které mohou následovat za povinnými poli TCP hlavičky. Volitelná položka je záznam typu TLV (type-length-value), který se skládá z typu volitelné položky, délky volitelné položky a hodnoty. Bližší informace o struktuře a významu jednotlivých volitelných položek lze nalézt opět v [13].

Padding – podobně jako u IPv4 protokolu slouží tato položka k zarovnání TCP hlavičkám pokud není její délka dělitelná 32 bity. Zarovnání se skládá z nulových bitů.

2.4.2 User Datagram Protocol

Protokol UDP [11] se stará o přenos paketů bez zajištění spolehlivosti nebo správného pořadí došlých paketů. Jedná se nespojovaný a bezstavový protokol. Formát UDP hlavičky je uveden na obrázku 2.5.



Obrázek 2.5: Formát UDP hlavičky

Source Port (16 bitů) – číslo zdrojového portu.

Destination Port (16 bitů) – číslo cílového portu.

Length (16 bitů) – délka dat včetně hlavičky udávaná v bajtech.

Checksum (16 bitů) – kontrolní součet počítaný z některých položek IP hlavičky a celého UDP datagramu.

2.5 Identifikace síťových toků

Důležitou funkcí každého systému stavového zpracování síťových toků je identifikace těchto toků. Jako „tok“ můžeme označit určitou podmnožinu síťového provozu, například komunikaci mezi dvěma procesy dvou koncových uzlů sítě, všechny pakety vyměněné mezi dvěma uzly bez ohledu na komunikující proces nebo všechny pakety putující z určité podsítě do jiné. U všech těchto případů je nutné vhodným způsobem určit, do kterého toku se má příchozí paket zařadit. S tím souvisí i úkol nalezení odpovídajícího kontextu datového toku v paměti. Každému toku je tedy třeba přidělit správný identifikátor, kterým bude označen příchozí paket. Nejčastější cesta, jak tento identifikátor pro příchozí paket zajistit, je výpočet hash hodnoty vybraným algoritmem nad zvolenými položkami hlaviček paketu. Zvolený algoritmus by měl dosahovat takových vlastností, aby nedocházelo k příliš častým kolizím nebo špatnému využití rozsahu možných identifikátorů. Zdrojové hlavičky pro hash funkci se mohou nacházet v různých vrstvách paketu, nejčastěji se jedná o linkovou, síťovou a transportní vrstvu.

Jako příklad uvedu výpočet identifikátoru pro datový tok mezi dvěma komunikujícími procesy dvou koncových uzlů sítě. Zde by se hash funkce počítala nad položkami *zdrojová* a *cílová IP adresa* z hlavičky síťové vrstvy a ze *zdrojového* a *cílového portu* z hlavičky transportní vrstvy.

Každá uživatelská aplikace může pro svoje zpracování vyžadovat jiný způsob identifikace toku (jiné položky hlaviček paketu), nad kterými bude vypočítán identifikátor. Podpora plně volitelných parametrů hash funkce je nutná pro dosažení dostatečné flexibility platformy s ohledem na využití v různých na síťových aplikacích.

Kapitola 3

Současné přístupy ke stavovému zpracování toků

Síťové zařízení provádějící analýzu datových toků může být realizováno pomocí osobního počítače, který obsahuje alespoň jednu síťovou kartu, a odpovídajícího programového vybavení. Takto založená síťová řešení však mají několik omezení, která zabraňují jejich efektivnímu využití v multigigabitových sítích. Jejich propustnost je velmi závislá na výkonu klíčových komponent osobního počítače, obzvláště procesoru a systémové sběrnice. Tato řešení jsou většinou schopna zpracovat datový tok o omezené propustnosti (sta Mb/s, maximálně jednotky Gb/s). Mezi softwarové systémy umožňující analýzu toků patří např. Bro [2] nebo Snort [15]. Oba systémy jsou zaměřeny na oblast bezpečnosti v počítačových sítích a způsoby detekce útoků.

Z důvodu nedostatečného výkonu osobních počítačů a jejich procesorů jsou výkonná síťová zařízení zpravidla realizována pomocí specializovaného technického vybavení, které je schopné zpracovat data na vysokých propustnostech. V poslední době vzrůstá důraz na vývoj architektur na bázi *System-on-a-chip* (SoC). Tyto architektury mohou být realizovány pomocí aplikačně-specifických obvodů (ASIC) nebo programovatelných hradlových polí (FPGA). Vyvíjená zařízení se zpravidla skládají z množiny modulů – IP jader. Tento způsob vývoje podporuje znovupoužitelnost vlastního zdrojového kódu nebo nákup komerčních IP jader. Celkově poskytuje efektivnější vývoj než při návrhu klasických fyzických obvodů.

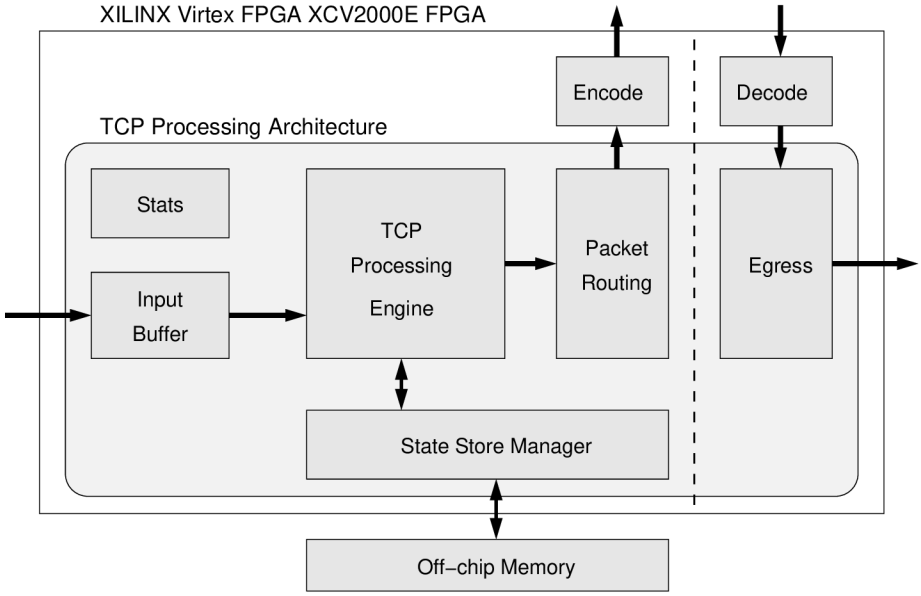
Aktuální situace dostupných architektur pro stavové zpracování toků je velmi dobře popsána v [10]. Podle této publikace není v současnosti, i přes svůj velký význam, vyvinuto mnoho zařízení schopných stavového zpracování toků na multigigabitových rychlostech. Publikací věnující se této problematice je také nedostatek. Přitom v oblasti bezpečnosti je stavové zpracování pro mnoho aplikací jedním ze základních předpokladů. V následujících částech bude popsáno několik důležitých prací zabývajících se stavovým zpracováním, uvedeny jejich přednosti a nedostatky a nakonec budou porovnány s navrhovanou platformou.

3.1 TCP-Processor

První publikací zaměřenou na stavové zpracování toků je příspěvek D. V. Schuehlera a J. W. Lockwooda. Jejich práce s názvem *A Modular System for FPGA-Based TCP Flow Processing in High-Speed Networks* (Modulární systém pro stavové zpracování TCP toků ve vysokorychlostních sítích) [14] předkládá zajímavý přístup k této problematice. Zabývá

se návrhem a implementací síťového zařízení pro TCP Reassembling (rekonstrukce TCP datového toku). Publikované zařízení nazvané TCP-Processor poskytuje uživatelské aplikaci prostředky ke zpracování TCP toků, zajišťuje ukládání a načítání stavové informace a poskytuje statistické údaje ke zpracovávaným datům.

Navržená a implementovaná jednotka je schopná zpracovat vstupní datový tok na rychlosti 2,5 Gb/s s maximálním počtem 8 milionů aktivních TCP toků. Uživatelské aplikaci je dáno rozhraní se sekvenčním přístupem (nelze k datům přistupovat náhodně), s rozlišením jednotlivých částí paketu (IP hlavička, TCP hlavička a TCP payload). Blokové schéma architektury TCP-Processoru je předvedeno na obrázku 3.1.



Obrázek 3.1: Architektura TCP-Processoru

Architektura TCP-Processoru se skládá z šesti oddělených komponent: *Input Buffer*, *TCP Processing Engine*, *State Store Manager*, *Packet Routing*, *Egress* a *Statistics Module*. Kromě těchto šesti hlavních komponent jsou ve schématu zakresleny i bloky *Encode* a *Decode*. Ty umožňují rozložení zpracování paketů na více FPGA. Tato vlastnost zde byla implementována pro snadnější použití rozsáhlého uživatelském zpracování paketů, které by nemohlo být umístěno na jeden FPGA čip s TCP-Processorem z důvodu nedostatku zdrojů. Přerušovanou čarou je ve schématu naznačeno možné rozdělení funkcionality do více FPGA. Komunikaci a přenos dat mezi jednotlivými FPGA zajišťují komponenty *Encode* a *Decode*. Datová cesta zpracovávaných paketů je vyznačena silnými černými šipkami. Tenčí šipky značí komunikaci mezi pomocnými komponentami.

Data vstupují do TCP-Processoru do komponenty *Input Buffer*, která má za úkol ukládat pakety do vyrovnávacích pamětí, pokud jsou následující komponenty zaneprázdněné zpracováním předchozích paketů. Tato zpoždění jsou většinou způsobena klientskou aplikací, která v některých situacích není schopna uplně zpracovat vstupní datový tok. Klíčovou jednotkou celého zařízení je následující jednotka *TCP Processing Engine*. Zde je provedena identifikace paketu a za pomoci komponenty *State Store Manager* je načtena odpovídající stavová informace. Po jejím načtení proběhne kontrola, zda přišel správný paket v pořadí (řídí se položkou *Sequence Number* TCP hlavičky). Po úspěšné kontrole je TCP paket zpra-

cován. Proběhne validace kontrolního součtu v TCP hlavičce a zapsání upravené stavové informace zpět do *State Store Manager*. Jednotka *Packet Routing* provádí transformaci vnitřních signálů TCP-Processoru na klientské rozhraní. Konečné napojení na uživatelské zpracování zajišťuje jednotka *Egress*.

Velmi důležitá komponenta celé architektury je výše zmíněný *State Store Manager*. Stará se o komunikaci s externí pamětí a o vyčítání a zpětné ukládání aktualizované stavové informace. Komponenta je vysoce optimalizovaná, protože paměťové operace mají vysoký vliv na propustnost každého systému pro stavové zpracování dat. Poslední komponentou je statistický modul, který sbírá užitečné informace při zpracování paketů jako je počet aktivních TCP spojení, počet ukončených spojení nebo množství zpracovaných dat.

3.2 TCP Stream Reassembly

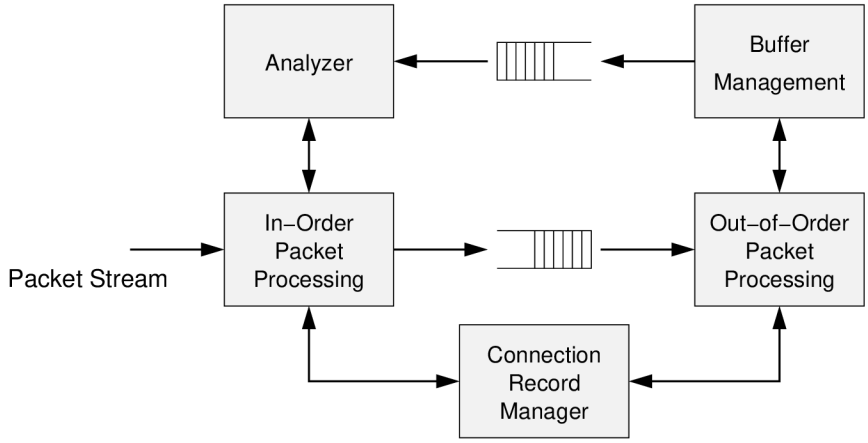
Článek S. Dharmapurikara a V. Paxsona nazvaný *Robust TCP Stream Reassembly In the Presence of Adversaries* (stavové zpracování TCP toků s přihlédnutím k útočníkům) [6] se zabývá analýzou živého toku na různých sítích a následným návrhem robustní architektury pro TCP Reassembling. Úkolem zařízení je úplná rekonstrukce TCP datových spojení z jednotlivých TCP paketů a poskytnutí výsledných souvislých toků dat uživatelským aplikacím. Na rozdíl od TCP-Processoru provádí tato architektura ukládání paketů mimo pořadí, čímž se podstatně zvyšuje efektivita přenosu dat po síti. Pokud by byly v zařízení pakety mimo pořadí zahazovány, vysílací strana by musela pakety odeslat znovu. Tím by došlo ke zpomalení komunikace mezi uzly.

Při návrhu brali autoři ohled na různé síťové útoky, aby způsobily co nejmenší poškození nebo degradaci výkonu síťového zařízení. Uvedu zde dva hlavní útoky, které byly v publikaci zkoumány a mohly by způsobit problémy v architektuře, která na ně není připravená. Jedná se o útoky typu *Denial-of-Service* (DoS), které mají za úkol omezit nebo úplně znemožnit funkci napadeného zařízení:

- **SYN flooding** – zasílání velkého množství SYN paketů (zahájení TCP spojení), které může vyčerpat zdroje napadeného zařízení zaplněním paměti záznamy o falešných TCP spojeních.
- **Záměrné zasílání paketů mimo pořadí** – tento útok je specifický hlavně pro systémy stavového zpracování dat, které provádí ukládání paketů mimo pořadí do pomocné paměti. Při tomto útoku jsou záměrně posílány TCP pakety mimo pořadí do napadeného zařízení s cílem přeplnit paměť určenou pro dočasné ukládání těchto paketů.

Byly detailně rozpracovány algoritmy pro kritické operace systému (zpracování příchozího paketu, ukládání paketů mimo pořadí, aktualizace záznamů o TCP spojení apod.), struktura záznamu o TCP spojení v paměti a předvedeno blokové schéma celé architektury, které je uvedeno na obrázku 3.2.

Skládá se z pěti hlavních funkčních bloků. *In-Order Packet Processing* zpracovává vstupní tok paketů v pořadí a předává je přímo uživatelské jednotce ke zpracování (*Analyzer*). Ta má opět k dispozici uživatelské rozhraní se sekvenčním přístupem k datům. Pokud dojde k příchodu TCP paketu mimo pořadí, jsou pakety daného TCP spojení přeposílány jednotce *Out-of-Order Packet Processing*, dokud nedojde k příchodu chybějícího paketu. Pro překlenutí případných prodlev při ukládání paketů mimo pořadí do paměti jsou mezi



Obrázek 3.2: Architektura TCP Reassembly

těmito jednotkami umístěny malé fronty. Obě jednotky pro zpracování paketů v pořadí i mimo pořadí ukládají záznamy o TCP spojeních do zvláštní paměti, kterou řídí *Connection Record Manager*.

Záznamy o TCP spojeních a pakety mimo pořadí jsou ukládány do rozdílných externích pamětí. To umožňuje lépe zřetězit zpracování paketů a zároveň využít výhody různých druhů externích pamětí pro jednotlivé úkoly. Pro ukládání paketů mimo pořadí autoři používají paměť DDR SDRAM, která je podle jejich výzkumů v současnosti jediná vhodná paměť pro tento účel, pokud má být zařízení dostatečně odolné proti síťovým útokům. Vlastní ukládání paketů mimo pořadí řeší komponenta *Buffer Manager*, ze které následně *Analyzer* vyčítá uložená data paketu.

3.3 Analýza architektur

Obě analyzované architektury se zabývají TCP Reassemblingem, což je pouze jedna z aplikací stavového zpracování toků. Cílová platforma, která je navrhovaná v této práci, by měla podporovat širší spektrum aplikací, ne pouze TCP Reassembling.

Při návrhu vyvíjené platformy byla analyzována struktura obou architektur a bylo nalezeno několik vlastností, které mohou být překážkou širšího využití zařízení:

- **Fixní identifikace datového toku** – v referenční architektuře byl tok vždy identifikován podle zdrojové a cílové IP adresy a portu. Zařízení tak může být využito pouze aplikacemi zpracovávajícími TCP toky.

- **Aplikaci není umožněno volné využití kontextu** – připojená aplikace by měla mít možnost zapsat do stavové informace vybraná data. V TCP-Processoru je obsah kontextu pevně dán. Pro větší flexibilitu by mělo zařízení umožnit také *volitelnou velikost kontextu*. S touto vlastností souvisí podpora operace *zpětného zápisu* upravené stavové informace do paměti. Ani jedna z těchto vlastností není v TCP-Processoru dostupná.

- **Volba externí paměti pro uložení kontextů** – uživatel by měl mít možnost volby externí paměti (např. mezi DDR SDRAM, SSRAM nebo QDR SSRAM). Proto by

měla mít jednotka provádějící zápis a vyčítání kontextů z paměti obecné rozhraní, aby se pro vybranou uživatelskou paměť mohla vytvořit tenká paměťově specifická vrstva. TCP-Processor fixně využívá pouze paměť typu SDRAM.

Dosud žádná publikace se nezabývala možností poskytnutí větší flexibility uživatelského rozhraní. Pro podporu širšího spektra aplikací chyběly následující vlastnosti:

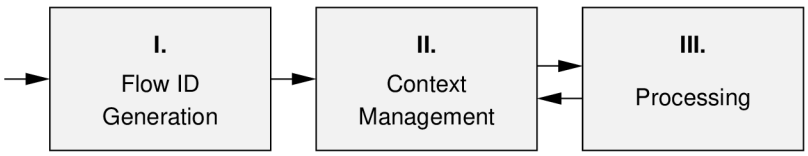
- **Volitelný počet uživatelských jednotek** – uživatelská aplikace může vyžadovat paralelní zpracování v několika procesních jednotkách. Tento požadavek může zpravidla nastat, pokud jedna uživatelská procesní jednotka není schopná zpracovat vstupní tok na plné propustnosti. Na vysokých propustnostech (10 Gb/s nebo 40 Gb/s) má zpracovávající zařízení většinou minimálně 4 procesní jednotky.
- **Datové rozhraní s náhodným přístupem** – některé aplikace mohou vyžadovat náhodný, nesekvenční, přístup k datům paketu. Mohou například analyzovat pouze několik bajtů z payloadu paketu, proto by pro ně sekvenční datové rozhraní znamenalo ztrátu výkonu.

Kapitola 4

Návrh a implementace platformy

4.1 Model stavového zpracování toků

Po analýze několika přístupů ke stavovému zpracování datových toků uvedených v kapitole 3 byl navržen model obecné platformy. Základní schéma modelu můžeme vidět na obrázku 4.1



Obrázek 4.1: Model stavového zpracování toků

Celé stavové zpracování je možné rozdělit do tří kroků:

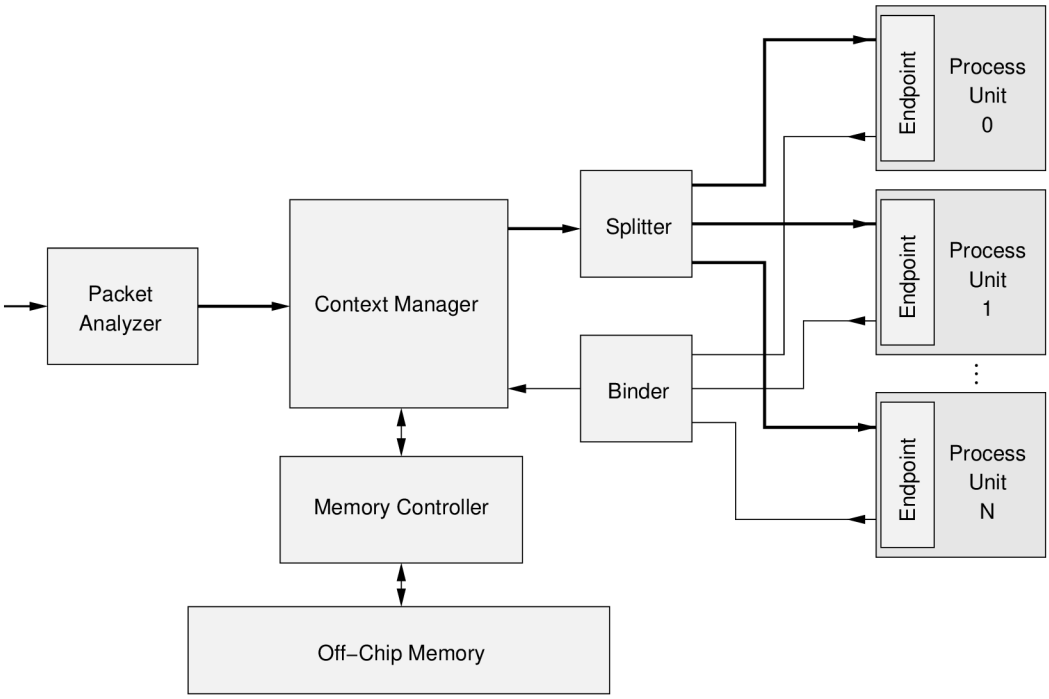
1. **Generování Flow ID** – pro každý příchozí paket je nutné nejprve provést identifikaci toku zmíněnou v kapitole 2.5). To obnáší výpočet identifikátoru na základě vybraných položek hlaviček paketů. Tento identifikátor bude v dalším textu označován jako *Flow ID*, které je také použito na lokalizaci odpovídajícího kontextu v externí paměti.

2. **Správa kontextu** – druhým krokem je nalezení odpovídajícího kontextu k příchozímu paketu a jeho vyčtení z paměti. Vyčítání a ukládání kontextu do paměti má největší vliv na propustnost celého systému, protože propustnost paměti je velmi často *úzkým místem* celé architektury. Je proto nutné snížit vhodnou optimalizací vliv externí paměti. Jak bylo zjištěno v kapitole 3.3, přístup k paměti by měl být obecný a pro jednotlivé druhy externích pamětí by měla být vytvořena tenká vrstva s konkrétním paměťovým rozhraním.

3. **Uživatelské zpracování** – poslední fází modelu je zpracování dat uživatelskou aplikací. Navržená platforma nabízí datové rozhraní s náhodným přístupem, které je využitelné pro více aplikací než rozhraní sekvenční. Uživateli je zpřístupněn payload paketu, zvolené hlavičky protokolů a odpovídající kontext. Po dokončení zpracování je kontext poslán zpět do Správy kontextu ke zpětnému uložení do paměti.

4.2 Architektura

V následujících kapitolách budou postupně probrány jednotlivé součásti navržené platformy, vysvětlena jejich funkce a u klíčových částí ukázána i jejich detailní struktura. Jako první bude předloženo blokové schéma platformy a postupně se přejde k detailnějším aspektům návrhu. Blokové schéma celé platformy pro stavové zpracování toků je uvedeno na obrázku 4.2.



Obrázek 4.2: Architektura platformy

Platforma se skládá ze šesti základních komponent. Vstupní tok paketů přijímá *Packet Analyzer*, který provede identifikaci paketu a extrakci vybraných uživatelských hlaviček. Tyto informace přijímá *Context Manager*, který je klíčovou jednotkou celého systému. Řídí vyčítaní odpovídajícího kontextu z paměti a jeho následné připojení k datům paketu. Komunikace s pamětí probíhá přes paměťově specifickou komponentu *Memory Controller*, která je připojena přímo na rozhraní vybrané externí paměti. Podle rozhodnutí *Context Manageru* vyšle *Splitter* paket s připojeným kontextem do odpovídajícího *Endpointu* ve vybrané procesní jednotce, kde je umístěna vlastní aplikace. *Endpoint* vytváří uživatelské rozhraní mezi platformou a vlastní aplikací a musí být vložen do každé procesní jednotky. Po dokončení uživatelského zpracování dat je aktualizovaný kontext zaslán zpět do *Context Manageru* pro zápis do paměti. Jednotlivé toky aktualizovaných kontextů jsou svázány do jediného toku pomocí *Binderu*.

V rámci této práce byly implementovány všechny komponenty kromě *Packet Analyzeru*. Ten lze seskládat z již hotových komponent, které byly vytvořeny v rámci projektu Liberouter [8]. Všechny komponenty jsou navrženy genericky, aby bylo možné pomocí jednoduché úpravy jejich parametrů změnit její vlastnosti jako je šířka zpracovávaného toku, velikosti front, pamětí a dalších parametrů. Tím se dosáhne dostatečné flexibility pro různé konfigurace platformy (verze pro 1 Gb/s, 10 Gb/s nebo jiné).

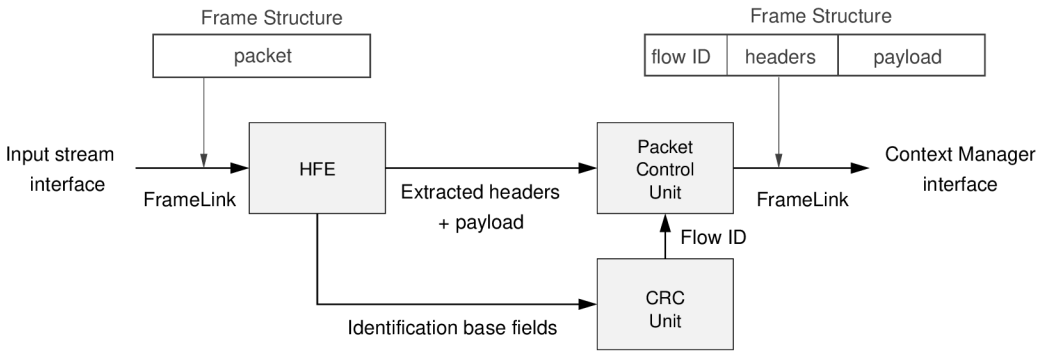
Komunikace mezi komponentami je zajištěna datovým protokolem s názvem *FrameLink*. Tento protokol byl vyvinut v rámci Liberouteru jako univerzální protokol pro jednotné propojení komponent. Vychází z protokolu LocalLink firmy Xilinx [16]. Protokol bude popsán pouze rámcově, podrobnou specifikaci lze nalézt ve výše uvedené publikaci. Základní datovou jednotkou FrameLinku je *rámec*. Ten může obsahovat jednu a více *částí*. Rámec tak může být rozdělen na tři části obsahující payload paketu, hlavičky paketu a kontext připojený k paketu nebo pouze jednu část s aktualizovaným kontextem. FrameLink má volitelnou datovou šířku, a proto je možné nastavit dostatečnou datovou propustnost pro jednotlivá spojení mezi bloky. Díky těmto vlastnostem protokolu je zajištěna dostatečná flexibilita celé architektury.

4.3 Packet Analyzer

Tato jednotka je první komponentou navržené platformy, ve které je zpracováván vstupní tok paketů. Jejím hlavním úkolem je příprava potřebných informací pro *Context Manager* a uživatelské zpracování. Pro každý příchozí paket musí *Packet Analyzer* provést následující kroky:

1. Analýza vstupního paketu
2. Extrakce zvolených hlaviček paketu
3. Identifikace – vygenerování Flow ID

První a druhý krok zajistí jednotka *Header Field Extractor* (HFE). Jedná se o procesor typu RISC, který na základě programu napsaném v assembleru zpracuje vstupní tok paketů a extrahuje vybraná pole hlaviček protokolů. Tato jednotka již byla dříve publikována jako bakalářská práce a implementována v rámci projektu Liberouter [9]. Na základě vybraných hlaviček protokolu paketu je vygenerováno Flow ID. Jako hash funkce byl vybrán algoritmus CRC (*Cyclic redundancy check*), který má dostatečně dobré výsledky pro rozptyl a kolize hash hodnot. Situace je znázorněna na obrázku 4.3.



Obrázek 4.3: Packet Analyzer

Výstupem bloku je FrameLinkový rámec obsahující dvě části. V první části je uloženo Flow ID a hlavičky extrahované pomocí HFE a v druhé payload paketu.

4.4 Context Manager

Tato jednotka je nejdůležitější a nejkomplexnější komponentou celého systému a na jejím výkonu závisí propustnost celé platformy. Proto byla tato komponenta velmi dlouho navrhována a optimalizována, aby dosáhla co nejlepších výsledků. Má čtyři základní úkoly:

- **Řízení vstupního toku** – *Context Manager* musí vhodně řídit příjem toku paketů z *Packet Analyzeru*. Pakety jsou dočasně ukládány do fronty, dokud není *Context Manager* připravený provést zpracování.
- **Rozhodnutí o způsobu zpracování paketu** – pro každý příchozí paket je nutné určit, do jaké procesní jednotky bude poslán a zda se pro něj má vyčítat kontext z paměti. Toto rozhodnutí souvisí se zajišťováním konzistence kontextu a je detailně popsáno v části 4.4.1.
- **Vyčtení aktuálního kontextu** – k příchozím paketům, pokud není stanoveno jinak, musí *Context Manager* vyčíst odpovídající kontext, který je lokalizován v externí paměti pomocí Flow ID. Tato operace má díky závislosti na rychlosti externí paměti velký vliv na propustnost celé platformy. Proto je *Context Manager* navržen tak, aby se propustnost paměti využila co nejvíce a nedocházelo ke zbytečným prodlevám při čekání na zpracování dalšího paketu. Požadavky na čtení z paměti se předávají *Memory Controlleru*.
- **Zpracování aktualizovaných kontextů** – z procesních jednotek přichází do *Context Manageru* tok aktualizovaných kontextů, které musejí být zapsány do externí paměti. Tato operace má větší prioritu než čtení kontextů k novým paketům, aby nedošlo k nadměrnému stárnutí aktualizovaných stavových informací při čekání na uvolnění paměti.

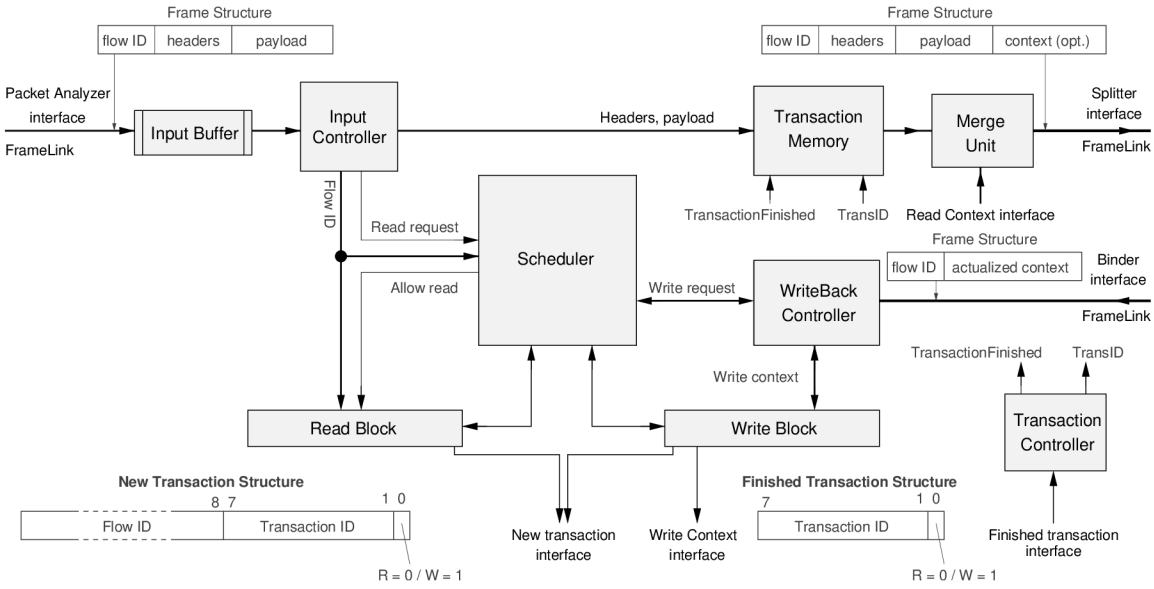
Context Manager je rozložen na množinu funkčních bloků, jejichž funkce bude dále vysvětlena. Detailní popis struktury bloků není pro pochopení funkce *Context Manageru* nezbytný a přesahoval by rozsah této publikace. Blokové schéma je uvedeno na obrázku 4.4.

Příchod nového paketu

Příchozí pakety jsou ukládány ve vstupní vyrovnávací paměti *Input Buffer*, dokud není *Context Manager* připraven je přijmout. Po přijetí je paket zpracován *Input Controllerem*, který extrahuje ze začátku FrameLinkového rámce hodnotu Flow ID, a zadá *Scheduleru* požadavek na čtení kontextu z paměti.

Scheduler je klíčovou jednotkou celého *Context Manageru*, která rozhoduje o příchozích paketech a jak s nimi bude naloženo. Pro zajištění konzistence paketu je v některých případech nutné použít aktuální kontext uložený v některé z procesních jednotek namísto kontextu uloženého v paměti. Základem *Scheduleru* je množina hlídacích jednotek (*Guard Unit*), každé procesní jednotce je přiřazena jedna z nich. Hlídací jednotky monitorují rozpracované pakety v procesních jednotkách a poskytují možnost kontroly, zda není v některé z procesních jednotek zjištěn další paket ze stejného toku. V kladném případě *Scheduler* nepovolí čtení kontextu z paměti, ale paket bude směrován přímo do vybrané procesní jednotky.

V druhém případě je zadána nová čtecí transakce externí paměti, kterou vygeneruje jednotka *Read Block*. Následně je paket odeslán do paměti transakcí (*Transaction Memory*),



Obrázek 4.4: Context Manager

kde je uložen do té doby, dokud není dokončeno čtení kontextu z paměti a kontext je připraven k odeslání. V prvním případě, kdy čtení kontextu z paměti nebylo nutné, nemusí paket v *Transaction Memory* čekat na přečtení kontextu a může být poslán přímo na výstup *Context Manageru*.

Ukládání paketů čekajících na přečtení kontextu z paměti do *Transaction Memory* umožňuje zpracování dalších paketů a generování požadavků do paměti. Tím se zlepšuje využití výkonu paměti, což je obzvláště důležité u pamětí s vysokou latencí, jako je SDRAM. Po uvolnění z *Transaction Memory* je paket předán spojovací jednotce (*Merge Unit*). Ta k rámci s daty paketu připojí kontext vyčtený z paměti (pokud bylo paketu čtení kontextu z paměti povoleno). Struktura výsledného FrameLinkového rámce je znázorněna na obrázku 4.4.

Zpracování aktualizovaného kontextu

Druhou důležitou funkcí *Context Manageru* je příjem aktualizovaných kontextů a jejich zpracování. Tato operace má vyšší prioritu než zpracování nového paketu. FrameLinkový rámec s aktualizovaným kontextem obsahuje pouze Flow ID a samotný kontext. Příjem těchto rámců řídí *WriteBack Controller*, který po příchodu rámce zadá *Scheduleru* požadavek na zapsání kontextu do paměti. Pokud se jedná o kontext jediného paketu datového toku v systému, je zápis povolen. V tom případě *Write Block* vygeneruje transakci zápisu do paměti a kontext je odeslán k zápisu. Pokud je v systému rozpracovaný další paket stejného datového toku, zápis do paměti není povolen a data jsou zahozena.

4.4.1 Zajištění konzistence

Při návrhu jednotky *Context Manager* bylo nutné zajistit, aby v procesních jednotkách byl vždy použit platný kontext (konzistenci kontextu). K porušení konzistence by mohlo dojít, pokud platforma přijme dva pakety ze stejného toku, které mají mezi sebou velmi malý

časový rozestup. V tomto případě by byl k prvnímu paketu vyčten platný kontext a došlo by ke správnému zpracování v procesní jednotce. Druhému paketu by ale byl z paměti vyčten zastaralý kontext, protože stejný kontext byl právě aktualizován v některé z procesních jednotek a nebyl ještě uložen v externí paměti.

Z popisu problému vyplývá, že *Context Manager* musí mít k dispozici informaci, zda je aktuální kontext k příchozímu paketu v externí paměti, nebo byl již vyčten a je aktualizován procesní jednotkou. Při znalosti těchto údajů lze konzistenci zajistit posláním paketu do té procesní jednotky, kde je uložen aktuální kontext, a nevyčítat stavovou informaci z paměti. Z tohoto důvodu je nutné v každé procesní jednotce použít paměť s dostatečným počtem buněk pro uložení kontextů. Buněk by mělo být právě tolik, jako je maximální počet rozpracovaných paketů mezi vyčtením kontextu z paměti, jeho zpracováním v procesní jednotce a uložením zpět do paměti. Tuto konstantu nazveme *maxPackets*. Konstanta závisí na konfiguraci platformy, ale ve většině případů by neměla přesáhnout číslo 10.

Context Manager může s buňkami v procesních jednotkách volně nakládat. Pro každou buňku musí existovat záznam, zda je obsazená aktualizovaným kontextem, nebo zda je volná. Dále se na jednu buňku může odkazovat více než jeden paket. K tomu dojde například v situaci popsané v prvním odstavci této podkapitoly. Obecně se může na buňku odkazovat 0 až *maxPackets* paketů. Všechny tyto pakety pocházejí ze stejného datového toku, a proto se také odkazují na kontext ve stejné buňce. Pro lepší vysvětlení práce s buňkami jsou zde uvedeny možné situace a jak se u nich bude postupovat:

- **První paket k danému toku** – protože se jedná o první paket v systému, k paketu je vyčten aktuální kontext z externí paměti. Pro zpracování se vybere libovolná procesní jednotka a kontext se uloží do první volné buňky, na kterou se neodkazuje žádný paket. Informace o vybrané buňce a procesní jednotce se uloží v *Context Manageru* pro pozdější využití. Čítač odkazů na vybranou buňku se zvýší na hodnotu 1 (na buňku se odkazuje právě jeden paket).

- **Další paket k danému toku** – pokud *Context Manager* zjistí, že k danému toku je již alespoň jeden rozpracovaný paket v systému, nesmí použít kontext z externí paměti, ale aktuální kontext v některé z buněk. Paketu tedy přesně určí, do které procesní jednotky se má poslat a v které buňce v dané procesní jednotce je aktuální kontext uložen. Hodnota čítače odkazů na buňku se zvýší o jedna.

- **Dokončení zpracování paketu** – po zpracování paketu v procesní jednotce je do *Context Manageru* zpět zaslán upravený kontext. Pokud se jedná o poslední paket k danému toku v systému, je vytvořen požadavek na zápis do externí paměti a je snížena hodnota čítače odkazů na danou buňku. V opačném případě je pouze dekrementován čítač – zápis do paměti by byl zbytečný. Pro další příchozí pakety k datovému toku by byl přiřazen kontext uložený v procesní jednotce, ne z paměti.

Context Manager musí pro každý tok uchovávat následující informace:

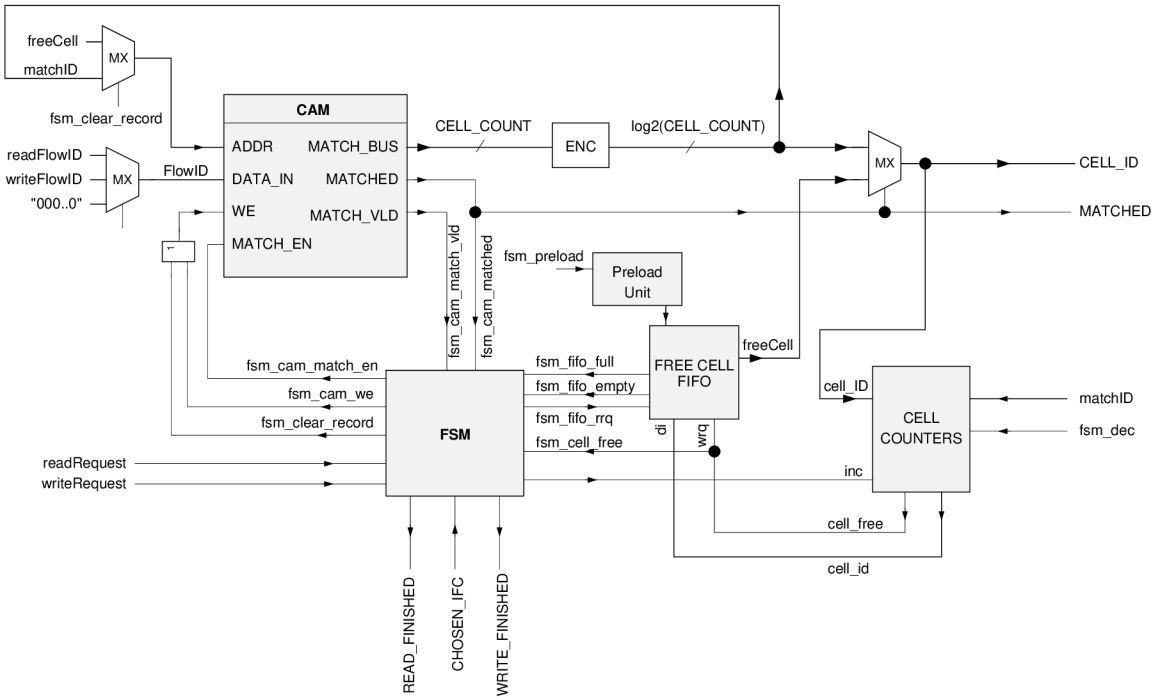
1. Kolik paketů je k danému toku v systému
2. Ve které procesní jednotce jsou tyto pakety zpracovávány
3. Kterou buňku v dané procesní jednotce využívají
4. Seznam volných buněk v procesních jednotkách

Pro každou informaci je nutné zvolit vhodný typ paměti, aby byla navržená hardwarová architektura dostatečně efektivní. Pro uložení druhé a třetí informace je nejvhodnější paměť adresovatelná obsahem (*Content-addressable Memory* – CAM). CAM má dvě hlavní funkce: ukládání záznamů pevné šířky na určitou adresu a vyhledávání záznamů. Při vyhledávání lze CAM předložit určitý záznam a pokud již byl do CAM někdy uložen, je vrácena jeho adresa. Vyhledání trvá konstantní dobu, u nejrychlejších implementací paměti pouze 1 takt.

V navržené platformě je pro každou procesní jednotku použita jedna CAM. Do paměti se ukládají Flow ID nových paketů na adresu, která je totožná s číslem vybrané buňky pro uložení kontextu ve vybrané procesní jednotce. Počty paketů odkazující se na jednotlivé buňky (první bod seznamu) jsou pro každou procesní jednotku uloženy v poli čítačů. Seznam volných buněk je udržován v paměti FIFO.

4.4.2 Guard Unit

Hlídací jednotka je klíčová pro celý *Context Manager*. Na základě informací v ní uložených se rozhoduje o způsobu zpracování příchozích paketů. Každé procesní jednotce je přiřazena jedna hlídací jednotka. Architektura *Guard Unit* je předvedena na obrázku 4.5.



Obrázek 4.5: Guard Unit

Na obrázku jsou znázorněny všechny paměti, které byly zmíněny v předchozí kapitole. Jednotka je řízena pomocí stavového automatu (FSM), který provádí jednotlivé kroky při rozhodování o zpracování paketů. Je nutné připomenout, že pro každou procesní jednotku je přiřazena právě jedna hlídací jednotka.

Při příchodu nového paketu z datového toku, který zatím není zpracováván v žádné z procesních jednotek, je vybrána procesní jednotka podle zvoleného algoritmu (náhodně, round robin nebo podle vytížení). V platformě bylo implementováno rozhodování podle vytížení procesních jednotek. Paket je tedy směrován do té procesní jednotky, která zpra-

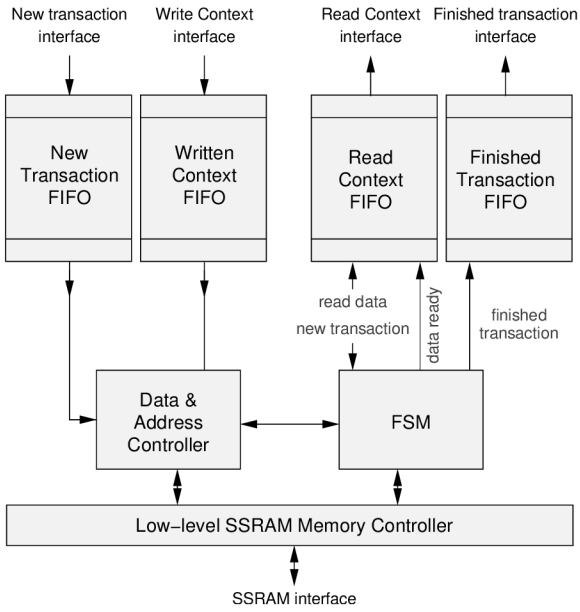
covává nejméně paketů. Hlídací jednotce odpovídající nejméně vytížené procesní jednotce je nastaven signál *CHOSEN_IFC*.

Při příchodu nového paketu je aktivován signál *readRequest*. Následně se v CAM podle Flow ID paketu vyhledá, zda není v hlídané procesní jednotce další paket ze stejného datového toku. Pokud je odpověď CAM kladná (*MATCHED*) je zvýšena hodnota odpovídajícího čítače v *Cell Counters* a paketu je přiřazena buňka zjištěná v CAM. V případě, že Flow ID nebylo v žádné hlídací jednotce nalezeno a paket je směrován do hlídané procesní jednotky, je paketu přiřazena volná buňka z fronty, zvýšen čítač této buňky a do CAM je zapsán záznam o novém paketu. Informace o vybrání dané hlídací jednotky k zapsání paketu je poskytnuta výše zmíněným signálem *CHOSEN_IFC*.

Žádosti o zápis aktualizovaného kontextu do paměti zpracovává také hlídací jednotka. Při příchodu aktualizovaného kontextu je nastaven signál *writeRequest*. Prvním krokem je dekrementace čítače odkazů na buňku, ve které byl kontext uložen. Pokud není na buňku žádný další odkaz (čítač je roven nule), je povolen zápis kontextu do paměti a z CAM je smazán záznam o tomto datovém toku.

4.5 Memory Controller

Řídící jednotka paměti (*Memory Controller*) se stará o transformaci obecného paměťového rozhraní *Context Manageru* na specifické rozhraní uživatelské paměti. Řídí správné zápisy aktualizovaných kontextů a čtení uložených kontextů z paměti. Pro účely této práce byl implementován *Memory Controller* pro statickou paměť SSRAM. Platformu je samozřejmě možné rozšířit o další typy *Memory Controllerů*, např. pro dynamickou paměť SDRAM nebo jiné druhy statických pamětí.



Obrázek 4.6: Řídící jednotka paměti SSRAM

Blokové schéma implementované jednotky lze nalézt na obrázku 4.6. V horní polovině schématu jsou znázorněny fronty pro příchozí požadavky čtení/zápisů do paměti. Ukládané

nebo čtené kontexty jsou taktéž ukládány do front. Po dokončení čtecí/zápisové transakce je *Context Manageru* zaslán záznam o vykonané transakci.

Vlastní provádění příchozích transakcí řídí stavový automat, který kontroluje připravenost externí paměti k operaci a volné místo ve výstupních frontách. Automat komunikuje přímo s nízkoúrovňovým kontrolérem paměti SSRAM, který byl již dříve implementován v rámci projektu Liberouter.

Pro lokalizaci kontextu v paměti se využívá Flow ID. V závislosti na velikosti paměti a kontextu se použije určitý počet nejvyšších bitů Flow ID jako adresa do paměti.Tuto transformaci provádí *Data & Address Controller*. Z uvedeného způsobu adresace vyplývá, že i dvě různá Flow ID, lišící se pouze ve spodních bitech, mohou adresovat stejný kontext v paměti. Tuto událost musí uživatelská aplikace detekovat, např. uložením celého Flow ID do kontextu a následné kontroly v procesní jednotce. Obecně platí, že čím je nastavena větší velikost kontextu a menší kapacita externí paměti, tím častěji bude docházet k těmto kolizím.

4.6 Splitter a Binder

Obě komponenty jsou v platformě využity pro flexibilní manipulaci s datovými toky. *Splitter* řídí rozdělování toku paketů do jednotlivých procesních jednotek. Vlastní informaci o tom, do které procesní jednotky a buňky se má paket odeslat, vkládá *Context Manager* do prvního slova FrameLinkového rámce. *Splitter* při příchodu nového paketu tuto informaci přečte a přepošle celý rámec s daty paketu do vybrané procesní jednotky.

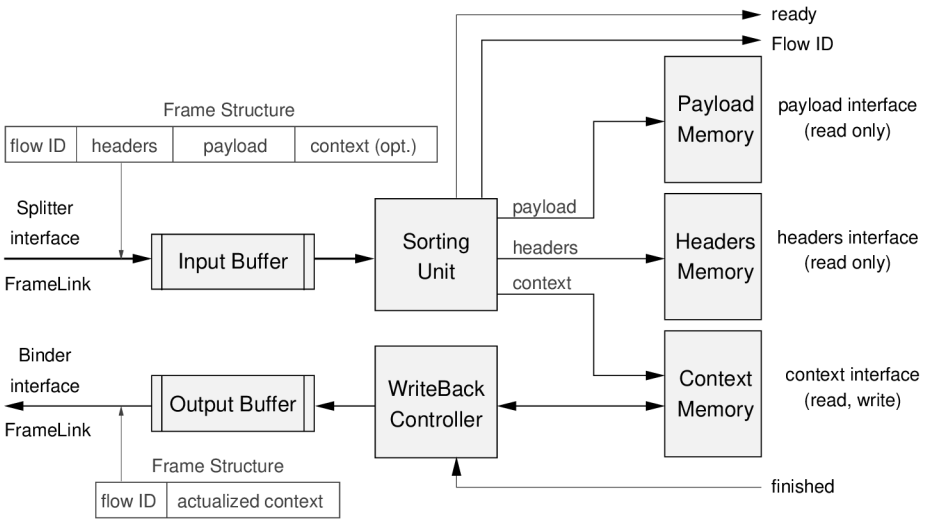
Binder, podobně jako *Splitter*, slouží pro komunikaci mezi *Context Managerem* a uživatelskými procesními jednotkami s *Endpointy*. *Binder* svazuje toky aktualizovaných kontextů z *Endpointů* do jediného toku. Oproti *Splitteru* podporuje volitelnou šířku vstupních toků i šířku výstupního toku.

4.7 Endpoint

Tato komponenta implementuje uživatelské rozhraní mezi aplikací a implementovanou platformou. Rozhraní bylo navrženo co nejjednodušší pro snadné a efektivní použití platformy. Uživatel má možnost rozhraní s náhodným přístupem k datům (rozhraní typu připojení na paměť). Sekvenční přístup by mohl některé aplikace, které analyzují jen některá data paketu, zdržovat a ubírat tak na efektivitě platformy.

Rozhraní s náhodným přístupem je náročnější na implementaci a na obsazené zdroje FPGA čipu. Je nutné použít paměti, které budou svojí kapacitou postačovat na několik uložených paketů. Ukládání více paketů je nutné z hlediska plného využití výkonu aplikace. Zatímco se zpracovávají data jednoho paketu, do *Endpointu* se musí nahrávat další paket, aby po dokončení zpracování prvního paketu byl již druhý k dispozici. V *Endpointu* je tedy třeba mít paměti pro 2 a více paketů a jejich hlaviček. Kromě těchto pamětí je zde použita paměť pro uložení kontextů s větším počtem buněk. Postup výpočtu minimálního počtu buněk byl vysvětlen v kapitole 4.4.1.

Blokové schéma *Endpointu* je zachyceno na obrázku 4.7. Vstupem je datový tok ze *Splitteru*, potažmo *Context Manageru*, který obsahuje Flow ID paketu, jeho hlavičky, payload a volitelně i kontext. V některých případech v příchozím rámci není kontext a je použit aktuální kontext v *Endpointu*. Vstupní data paketu jsou zpracována třídící jednotkou (*Sorting Unit*), která rozdělí payload, hlavičky a kontext do jednotlivých koncových



Obrázek 4.7: Endpoint

pamětí. Po zpracování všech dat uživatelskou aplikací převezme *WriteBack Controller* kontrolu nad pamětí kontextů a vyšle aktuální kontext do *Binderu*, který toky kontextů ze všech *Endpointů* skládá do jediného toku do *Context Manageru*.

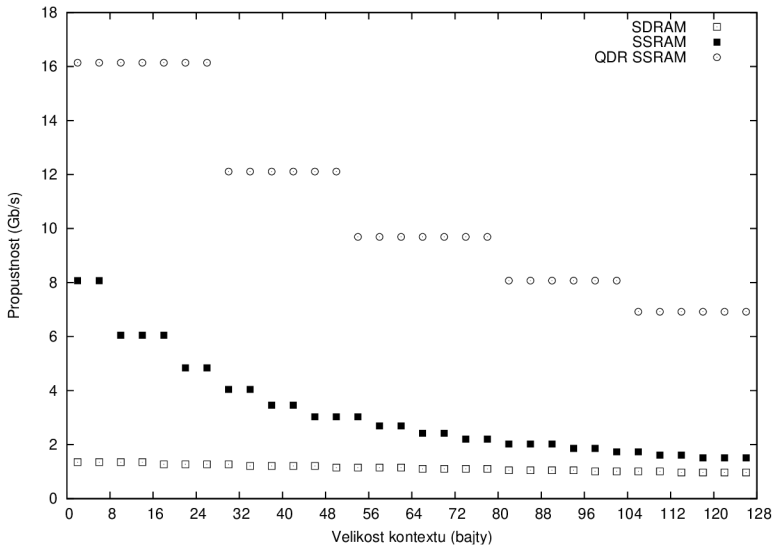
Připravenost všech dat je uživateli oznámena pomocí signálu *ready*. Společně s tímto signálem se vystaví i Flow ID paketu. To může být využito například v aplikacích provádějících další stavové zpracování. Ukončení zpracování dat oznámí uživatel signálem *finished*.

Kapitola 5

Dosažené výsledky

Při návrhu platformy byly prozkoumány její možnosti a analyzovány její důležité vlastnosti, které je nutné zvážit při vývoji aplikace nad touto platformou. Jedním z nejdůležitějších parametrů je propustnost, která je obzvláště důležitá pro aplikace ve vysokorychlostních sítích. Díky možnosti volby šířky datové sběrnice mezi jednotlivými komponentami je pro ně možné nastavit požadovanou propustnost. Tuto vlastnost zajišťuje protokol FrameLink zmíněný v kapitole 4.2. Zařízení je tak limitováno pouze propustností procesních jednotek a použitou externí pamětí. Propustnost procesních jednotek je možné zvýšit paralelním zpracováním dat v potřebném množství jednotek. Samozřejmě je třeba brát ohled na dostupné zdroje na cílovém FPGA čipu.

Důležitou součástí platformy, kterou lze optimalizovat pouze v omezeném měřítku, je paměť na uložené kontexty. Paměť má fixní parametry jako je kapacita, propustnost nebo latence, které jsou dány výrobcem a použitou technologií při výrobě. Proto byly vypočítány minimální propustnosti pro několik typů paměti. Ve výpočetním modelu se vstupní tok skládal pouze z nejkratších paketů, a proto musela platforma provádět mnoho čtení a zápisů do paměti. Z tohoto důvodu jsou vypočítané propustnosti minimální a pro skutečný síťový provoz by dosáhla platforma lepších výsledků. Výsledný graf je uveden na obrázku 5.1.



Obrázek 5.1: Minimální propustnost pro vybrané paměti

Propustnost byla vypočítána pro 3 druhy pamětí: dynamická paměť SDRAM a dva druhy statických pamětí: klasická SSRAM a rychlá dvouportová QDR SSRAM. Při výpočtu byly pro platformu použity hodiny s frekvencí 100 MHz. Parametry pamětí, na kterých byl založen výpočet, jsou uvedeny v tabulce 5.1.

Tabulka 5.1: Parametry pamětí pro výpočet propustnosti

	kapacita	latence	propustnost
	Mb	*hodinové takty*	*Bajtů / hodinový takt*
SDRAM	1024	17	16
SSRAM	2	2	9
QDR SSRAM	8	2	26

Z grafu 5.1 je patrné, že výsledná propustnost je závislá na délce kontextu, proto musí být množství informací ukládaných do kontextů v aplikaci dobře navrženo. Při výběru paměti do aplikace je třeba se rozhodnout mezi rychlostí paměti a kapacitou. Čím rychlejší paměť zvolíme, tím menší kapacitu bude většinou poskytovat. Menší kapacita vede k menšímu počtu možných kontextů a častějším kolizím mezi přiřazovanými kontexty, které mají negativní vliv na výkon platformy.

5.1 Využití zdrojů a dosažená frekvence

Vedle dosažitelné propustnosti je pro cílovou aplikaci rovněž důležité množství platformou obsazených zdrojů FPGA. Obsazené zdroje jsou nejvíce závislé na počtu procesních jednotek, protože pro každou procesní jednotku musí být vytvořena jedna poměrně rozsáhlá hlídací jednotka.

Nad implementovanou platformou byla provedena syntéza. V tabulce 5.2 jsou uvedeny výsledky syntézy několika verzí platformy. Jsou zde uvedeny hodnoty v jednotkách *Slices* a paměťových elementů *BlockRAM*. Dosažené výsledky jsou uspokojivé a vytvořená implementace poskytuje prostor pro další optimalizace.

Tabulka 5.2: Obsazení čipu FPGA

Architecture		1 proc. jednotka	2 proc. jednotky	4 proc. jednotky
1 Gb/s	Slice[a]	1126 (5%)	1952 (8%)	3613 (15%)
	BlockRAM[b]	21 (9%)	28 (12%)	40 (17%)
4 × 1 Gb/s	Slice	1120 (5%)	2002 (9%)	3767 (16%)
	BlockRAM	16 (7%)	22 (10%)	36 (16%)
10 Gb/s	Slice	1742 (7%)	2934 (12%)	5345 (23%)
	BlockRAM	32 (14%)	50 (22%)	84 (36%)

[a] Maximálně 23616 pro čip `xc2vp50`
[b] Maximálně 232 pro čip `xc2vp50`

Při syntéze byla pro všechny verze architektury dosažena pracovní frekvence minimálně 100 MHz, která je momentálně pracovní frekvencí všech aplikací vyvíjených v rámci projektu Liberouter.

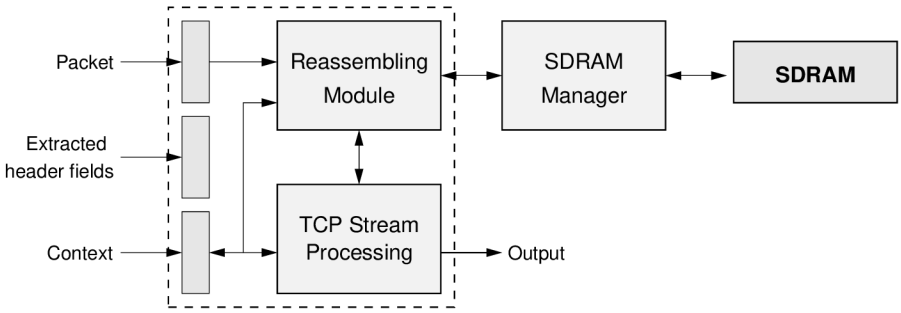
Kapitola 6

Aplikace

Navržená platforma pro stavové zpracování síťového provozu je velmi flexibilní z hlediska použitelnosti pro různé aplikace. Jednotlivé vlastnosti navržené architektury, kterými bylo dosaženo potřebné flexibility, byly zmíněny při popisu modelu a struktury platformy. V této kapitole budou popsána tři různá síťová zařízení, na kterých se široké využití platformy demonstruje.

6.1 TCP Reassembling

První aplikací je zařízení pro TCP Reassembling. Jeho úkolem je úplná rekonstrukce TCP datových spojení a poskytnutí výsledných souvislých toků dat uživatelským aplikacím. Zařízení musí vhodně zpracovávat pakety mimo pořadí, protože ve vstupním síťovém toku nemusí být TCP pakety uspořádané. Může docházet k předbíhání paketů, duplikacím nebo úplné ztrátě paketu [6]. Proto musí zařízení pakety mimo pořadí vhodně ukládat a uchovávat záznamy o aktivních TCP spojeních. Možné využití platformy pro zařízení implementující TCP Reassembling je uvedeno na obrázku 6.1.



Obrázek 6.1: Procesní jednotka pro TCP Reassembling

Identifikace paketu je v TCP Reassemblingu založena na zdrojové a cílové IP adrese a TCP portu. Do kontextu se uloží Flow ID pro kontrolu, zda nedošlo ke kolizi dvou hash hodnot, TCP pořadové číslo (*sequence number*) a pole adres do paměti SDRAM, kde se ukládají pakety mimo pořadí. Pole adres představuje *okno* pevné velikosti pro ukládání těchto paketů. Adresy jsou seřazeny podle pořadového čísla.

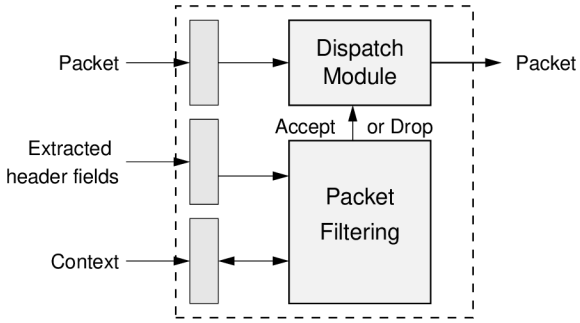
Při příchodu nového paketu proběhne kontrola jeho pořadového čísla. Po kontrole mohou nastat dvě situace:

- **Paket mimo pořadí** – tento případ nastává, pokud pořadové číslo paketu nesouhlasí s počátkem okna. Paket neobsahuje navazující data rozpracovaného toku a musí být dočasně uložen do paměti, dokud nemá systém k dispozici chybějící pakety. Pro uložení paketu je využita paměť SDRAM a do odpovídající položky v kontextu je vyplněna adresa uloženého paketu.
- **Paket v pořadí** – při rovnosti pořadového čísla paketu s počátkem okna je paket přeposlán ke zpracování uživateli. Také musí proběhnout kontrola, zda nejsou v paměti již uloženy další pakety k danému TCP spojení. Pokud se paketem v pořadí zacelí mezera v TCP toku, do uživatelského zpracování jsou odeslány i pakety uložené v SDRAM. Jejich adresy se zjistí v kontextu k TCP spojení.

Paměť SDRAM je pro účely této aplikace rozdělena na bloky pevné velikosti odpovídající maximální délce paketu. Správu bloků má na starosti jednotka *SDRAM Manager*. Vlastní kontrola paketů mimo pořadí, rozhodování o zpracování paketu je řešena v *Reassembling Module*. Jednotka *TCP Stream Processing* implementuje vlastní napojení na uživatelské zpracování.

6.2 Stavový firewall

Druhou uvedenou aplikací je stavový firewall implementovaný v FPGA. Jeho úkolem je rozhodnout o přijetí či zahození příchozích paketů na základě uživatelských pravidel. Nákres odpovídající procesní jednotky je uveden na obrázku 6.2.

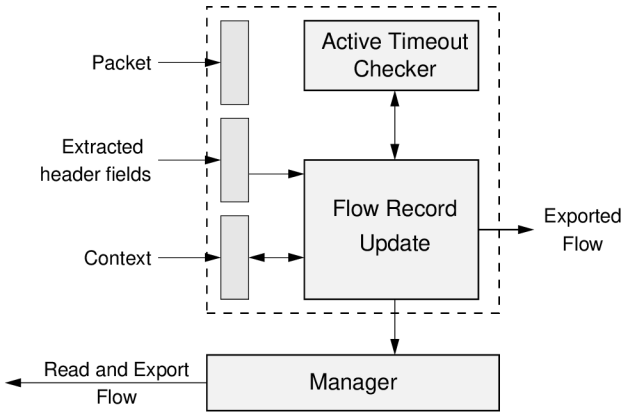


Obrázek 6.2: Procesní jednotka pro stavový firewall

Datový tok je identifikován stejný způsobem jako u aplikace TCP Reassembling, tzn. na základě zdrojové a cílové IP adresy a TCP portu. V kontextu poskytovaném k datovému toku je uložen stav spojení daného toku a rozhodnutí firewallu o přeposílání či zahazování daného toku. Jednotka *Packet Filtering* implementuje funkce klasifikace paketů a ukládání a načítání potřebných informací do kontextu. Rozhodnutí o paketu je předáno jednotce *Dispatch Module*, která provádí vlastní přeposlání nebo zahození paketu. Rozhodnutí se musí provést pro datové toky pouze jednou, při příchodu prvního paketu. Zpracování všech následujících paketů probíhá na základě uloženého rozhodnutí.

6.3 Sonda NetFlow

Třetí aplikací je zařízení monitorující vstupní tok na základě protokolu NetFlow [4] od firmy Cisco [3]. Zařízení poskytuje mnoho agregovaných informací IP tocích, které mohou být s výhodou použity pro správu sítě, při návrhu nové topologie sítě nebo detekci *DoS* útoků. Návrh procesní jednotky pro tuto aplikaci je uveden na obrázku 6.3.



Obrázek 6.3: Procesní jednotka pro sondu NetFlow

V kontextu k datovému toku jsou pro tuto aplikaci uloženy NetFlow záznamy přesně definované normou [4]. Při příchodu nového paketu jsou jednotkou *Flow Record Update* vypočítány nové hodnoty pro všechna pole v NetFlow záznamu na základě daných agregačních funkcí implementovaných v procesní jednotce. Pokud dojde k přetečení časového limitu spojení pro IP tok, NetFlow záznamy jsou odeslány z platformy. Tato přetečení jsou kontrolována jak při každém příchodu paketu do procesní jednotky, ale i periodicky pro všechny záznamy v paměti. Exportované záznamy jsou zachytávány NetFlow kolektorem, který je realizován pomocí programového vybavení počítače. Data uložená v kolektorech jsou následně uživateli přehledně prezentována v libovolné NetFlow aplikaci.

Kapitola 7

Závěr

Cílem této práce byl návrh a implementace flexibilní platformy pro stavové zpracování síťových toků v podobě množiny IP jader, které by byly snadno použitelné v širokém spektru síťových aplikací na bázi systém na čipu (SoC). Tento cíl byl splněn. Široké uplatnění navržené architektury bylo demonstrováno v kapitole 6 na příkladu tří klíčových aplikací. Mezi hlavní přednosti navržené platformy je možné uvést volné použití kontextu, jeho volitelnou délku, nezávislost na externí paměti pro uložení kontextu a podporu více procesních jednotek. Nad rámec zadání byla architektura rozšířena o zpracování libovolných toků, ne pouze TCP/IP toků.

Byla nastudována literatura na téma FPGA [17, 18] a jazyku VHDL [1] pro lepší pochopení cílového zařízení *Combo6X*, vyvinutého na projektu Liberouter [8]. Dále byly prostudovány odborné články zabývající se tématikou stavového zpracování toků [14, 6]. Informace nutné k hlubšímu pochopení vrstvového modelu ISO/OSI byly získány v odpovídajících RFC [11, 12, 13]. Načerpané znalosti a zkušenosti byly použity při psaní teoretického úvodu.

Při vývoji platformy byl kladen velký důraz na efektivní návrh, který by umožnil zpracování toků na co největší propustnosti. Závislost propustnosti na různých druzích pamětí byla analyzována v kapitole 5. Vytvořený návrh byl implementován v jazyce VHDL s ohledem na syntézu do programovatelného hradlového pole čipu Virtex-II Pro. Jednotlivá IP jádra byla implementována velmi genericky, aby je bylo možné pomocí jednoduché úpravy parametrů nasadit v různých prostředích a aplikacích.

Správná funkce výsledné implementace byla ověřena v testovacím prostředí v simulacích. Ve spolupráci s vývojovým týmem Intrusion Detection System na projektu Liberouter byla také ověřena vhodnost nasazení platformy pro zařízení detekce vzorů v síťovém provozu. Výsledky získané po syntéze implementace platformy byly předvedeny v kapitole 5. Míry obsazení čipu pro různé síťové architektury jsou uspokojivé, pracovní frekvence pro různé typy architektur překračovaly 100 MHz. Více informací lze nalézt ve výše zmíněné kapitole.

Současné výsledky práce mohou být dále zdokonalovány z hlediska dosažené propustnosti a obsazeného místa na čipu. Vývoj platformy bude dále pokračovat a platforma bude nasazena ve více síťových aplikacích, které prověří její vlastnosti.

Literatura

[1] Ashenden, P. J.: *The VHDL Cookbook*. červenec 1990, dokument dostupný na URL `http://tech-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf` (květen 2007).

[2] Bro: Webové stránky projektu Bro Intrusion Detection System.
URL http://bro-ids.org

[3] Cisco: Webové stránky společnosti Cisco.
URL http://www.cisco.org

[4] Claise, B.: *RFC 3954 - Cisco Systems NetFlow Services Export Version 9*. 2004, dokument dostupný na URL `http://rfc.net/rfc3954.txt` (květen 2007).

[5] Deering, S.; Hinden, R.: *RFC 2460 - Internet Protocol Version 6 (IPv6) Specification*. 1998, dokument dostupný na URL `http://rfc.net/rfc2460.txt` (květen 2007).

[6] Dharmapurikar, S.; Paxson, V.: Robust TCP stream reassembly in the presence of adversaries. In *Proceedings of USENIX Security*, 2005.

[7] IEEE: *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. IEEE, std 802.3 vydání, 2002, dokument dostupný na URL `http://standards.ieee.org/getieee802/download/802.3-2002.pdf` (květen 2007).

[8] Liberouter: Webové stránky projektu Liberouter.
URL http://www.liberouter.org

[9] Mikušek, P.: *Návrh a implementace procesní jednotky pro analýzu vstupních paketů*. bakalářská práce, FIT VUT v Brně, 2005.

[10] Paxson, V.; Asanovic, K.; Dharmapurikar, S.; aj.: Rethinking Hardware Support for Network Analysis and Intrusion Prevention. In *Proceedings of USENIX Hot Security*, 2006, s. 63–68.

[11] Postel, J.: *RFC 768 - User Datagram Protocol*. 1980, dokument dostupný na URL `http://rfc.net/rfc793.txt` (květen 2007).

[12] Postel, J.: *RFC 791 - Internet Protocol*. 1981, dokument dostupný na URL `http://rfc.net/rfc0791.txt` (květen 2007).

[13] Postel, J.: *RFC 793 - Transmission Control Protocol*. 1981, dokument dostupný na URL `http://rfc.net/rfc793.txt` (květen 2007).

[14] Schuehler, D. V.; Lockwood, J. W.: A Modular System for FPGA-Based TCP Flow Processing in High-Speed Networks. In *FPL*, 2004, s. 301–310.

[15] Snort: Webové stránky projektu Snort.
URL http://www.snort.org

[16] Xilinx: *LocalLink Interface Specification*. červenec 2005.
URL http://www.xilinx.com

[17] Xilinx: *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*. březen 2007, dokument dostupný na URL `http://direct.xilinx.com/bvdocs/userguides/ug012.pdf` (květen 2007).

[18] Xilinx: *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet*. březen 2007, dokument dostupný na URL `http://direct.xilinx.com/bvdocs/publications/ds083.pdf` (květen 2007).

Příloha A

Paměťové médium

K bakalářské práci je přiloženo paměťové médium (CD) obsahující elektronickou verzi technické zprávy, zdrojové soubory vyvinuté platformy pro stavové zpracování toků a programovou dokumentaci.