VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

STAVOVÝ FIREWALL V FPGA

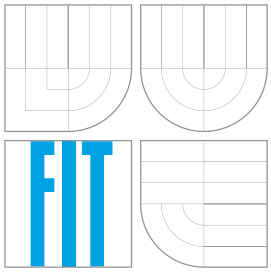
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN ŽIŽKA

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

STAVOVÝ FIREWALL V FPGA

STATEFUL FIREWALL FOR FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN ŽIŽKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VIKTOR PUŠ

BRNO 2012

Abstrakt

Tato práce popisuje analýzu požadavků, návrh a implementaci stavového filtrování paketů do již existujícího bezstavového firewallu. Zabývá se také testováním implementovaného systému. V úvodních dvou kapitolách popisuje vlastnosti vývojové platformy NetCope pro FPGA. Popisuje také princip činnosti firewallu, který zároveň slouží jako specifikace požadavků na stavový firewall. Poté popisuje detailní návrh na úpravy jednotlivých modulů existujícího firewallu a také návrh na vytvoření nových modulů. Zabývá se také implementací navržených modulů a otestováním jejich správné funkčnosti. Závěrem diskutuje současný stav práce a popisuje možná další rozšíření.

Abstract

This thesis describes the requirements analysis, design and implementation of stateful packet filtering to an existing stateless firewall. They also deals with testing of the implemented system. The first two chapters describe the properties NetCOPE development platform for FPGA. They also describes the principle of operation firewall, which also serves as a requirements specification for stateful firewall. Then describes the detailed design of individual modules to modify the existing firewall and the proposal for the creation of new modules. It also discusses the implementation of the proposed modules and testing for proper operation. Finally, it discuss the current state of the thesis and describes possible future expansion.

Klíčová slova

Stavový firewall, FPGA, VHDL, NetCOPE, filtrování paketů, bezpečnost

Keywords

Stateful firewall, FPGA, VHDL, NetCOPE, packet filtering, security

Citace

Martin Žižka: Stavový firewall v FPGA, diplomová práce, Brno, FIT VUT v Brně, 2012

Stavový firewall v FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Viktora Puše. Uvedl jsem všechny zdroje, ze kterých jsem čerpal.

.....
Martin Žižka
18. května 2012

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu této práce Ing. Viktoru Pušovi za poskytnuté rady a konzultace, bez nichž by tato práce nevznikla. Také bych chtěl poděkovat sdružení CESNET za poskytnutí hardwaru pro testování a v neposlední řadě rodině a přítelkyni za podporu při psaní této práce.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Zabezpečení počítačové sítě je v dnešní době důležitou součástí síťové komunikace. Zabraňují vytvoření neodhalené neautorizované komunikaci, během které může dojít ke ztrátě citlivých dat, přihlašovacích jmen a hesel či páchání trestné činnosti pod jménem oběti z nezabezpečené sítě, vytvoření tzv. botnetu či útoku Denial Of Servise na nezabezpečenou síť, který spočívá v nedostupnosti služby pro ostatní uživatele. Pro zabezpečení počítačové sítě bylo vytvořeno několik zařízení a nástrojů. Jedním ze zařízeních sloužících pro zabezpečení počítačové sítě je firewall.

Firewall slouží pro klasifikaci a filtrování jednotlivých paketů i celých datových toků mezi počítačovými sítěmi s různou úrovní důvěryhodnosti a zabezpečení. Pracuje s pravidly, která slouží pro definování povoleného nebo zakázaného stavu síťové komunikace mezi sítěmi, které od sebe odděluje. Historicky pravidla vždy zahrnovala zdrojovou a cílovou IP adresu, zdrojový a cílový port a typ komunikačního protokolu. Modernější firewally dokáží sledovat i stav komunikace, případně provádět stavovou inspekci běžně se vyskytujících protokolů i aplikační vrstvy a prvky systémů pro odhalení průniku (IDS).

V dnešní době dochází ke vzrůstání požadavků na rychlost síťové komunikace. Dnešní vysokorychlostní a páteřní sítě dosahují rychlostí 1 Gb/s až 10 Gb/s, přičemž vývoj sítí směřuje dále k vyšším rychlostem 40-100 Gb/s. Aby mohly být tyto požadavky realizovány, je třeba vyvíjet stále rychlejší a rychlejší síťové prvky. Zároveň je třeba stále dbát na monitorování a zabezpečení počítačové sítě.

Pro dosažení požadovaných rychlostí nestačí použít běžný počítač s programem, proto rychlé síťové prvky využívají specializovaný hardware. Ten zajišťuje dostatečnou výkonnost pro přenášení požadovanými rychlostmi. Navrhnout hardwarovou aplikaci se specifickou funkčností (ASIC) však vyžaduje nákladný vývoj a ověření správné funkčnosti před vytvořením samotného obvodu. Důraz je kladen hlavně na simulace, protože pokud se v architektuře vyskytuje chyba, tak je chybný celý obvod bez možnosti opravy. Proto se v poslední době při vývoji těchto aplikací sahá po technologii FPGA, což je technologie rekonfigurovatelného hardwaru. Tato technologie umožňuje rychlý vývoj hardwarově akcelerovaných aplikací s možností opravy odhalených chyb pomocí nahrání nové konfigurace obvodu.

Tato diplomová práce se zabývá analýzou požadavků, návrhem a implementací statového firewallu s využitím platformy pro vývoj síťových aplikací NetCOPE. V kapitole 2 je uveden popis technologie FPGA označující programovatelný hardware. FPGA čip tvoří základ karty rodiny COMBOv2, jejíž základní popis je také v této kapitole. Kapitola 3 se zabývá rozsáhlejším popisem platformy NetCOPE, na které je postaven vývoj existujícího bezstavového firewallu. Kapitola 4 popisuje činnost firewallů a jejich typy. Pro každý typ

uvádí, jak funguje a jeho výhody a nevýhody. Tato kapitola zároveň slouží jako analýza požadavků na stavový firewall, neboť popisuje jeho chování pro různé, často se vyskytující, protokoly.

Kapitola 5 obsahuje detailní návrh úprav existujících modulů a nově vytvářených modulů. U existujících modulů popisuje jejich funkci v rámci bezestavového firewallu a zároveň jejich modifikaci pro stavové filtrování. U nově vytvořených modulů uvádí princip činnosti, rozhraní modulu a připojení do existujících částí. Kapitola 6 seznamuje s implementací dle návrhu uvedeného v kapitole 5. Kapitola 7 popisuje způsob testování doimplementovaného stavového filtrování.

V závěrečné části je diskutován současný stav funkčnosti stavového firewallu a možnosti dalšího rozšíření, včetně jejich stručného návrhu.

Kapitola 2

Hardwarové vybavení

Tato kapitola seznamuje se základními principy obvodů FPGA (Field-programmable Gate Array). Dále seznamuje s existující hardwarovou kartou COMBOv2, která je osazena mimo jiné FPGA obvodem.

2.1 Technologie FPGA

Technologie FPGA tvoří základní prvek programovatelného hardwaru. Jedná se o technologii, která umožňuje pomocí integrovaných obvodů schopných měnit svou vnitřní konfiguraci na základě návrhu zařízení. Pomocí konfigurace je dosaženo požadovaného chování čipu. Informace k této podkapitole vycházejí z [13].

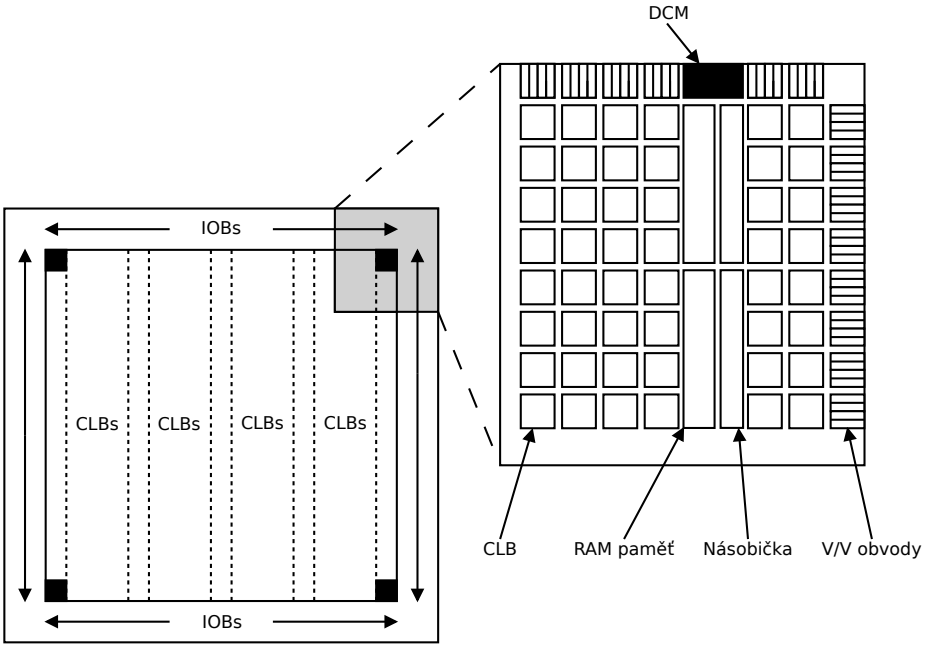
Tato technologie tvoří kompromis mezi technologií ASIC a použitím klasického procesoru, jaký lze nalézt např. v počítači. Technologie ASIC, nebo-li *Application Specific Integrated Circuid*, značí integrovaný obvod navržený speciálně pro konkrétní aplikaci. Kvůli velkým nákladům na výrobu masky pro vytvoření struktury obvodu se výroba ASIC obvodu vyplatí při milionových sériích. Oproti tomu univerzální procesor pro vestavěné systémy nabízí levnou výrobu a flexibilitu, ale nedosahuje požadovaných výkonů. Je to způsobeno režií při zpracování jednotlivých instrukcí[1] a sériovým prováděním instrukcí pro jednojádrový procesor. V dnešní době technologie FPGA nabízí téměř stejný výkon jako ASIC a díky snadné rekonfiguraci usnadňuje vývoj aplikace a snižuje cenu vývoje, jelikož není třeba vytvářet masku pro vytvoření struktury obvodu tak jako u ASIC obvodů.

Funkce FPGA čipu je dána konfigurací vnitřní struktury, proto je třeba přizpůsobit strukturu čipu tak, aby bylo možné rekonfiguraci provést. Obrázek 2.1 znázorňuje vnitřní strukturu FPGA čipu.

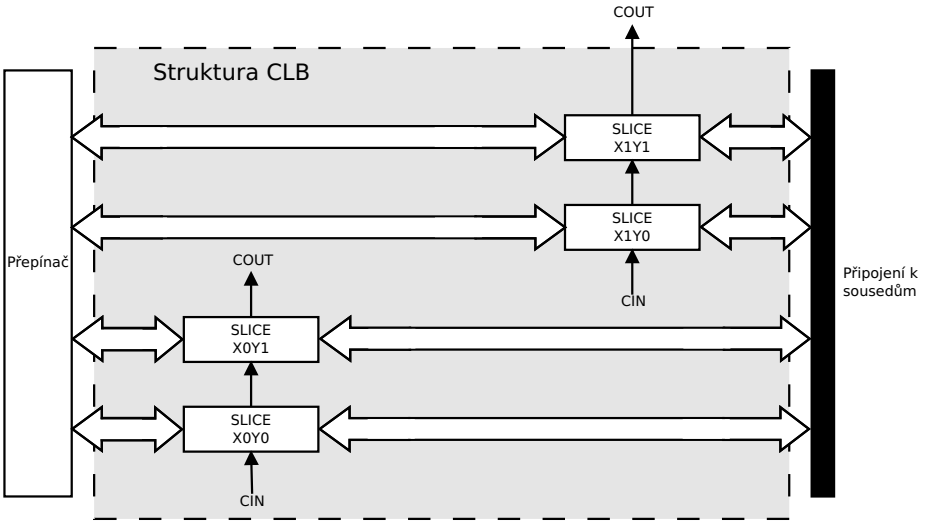
2.1.1 Programovatelné logické bloky

Funkci FPGA obvodu zajišťují CLB, nebo-li konfigurovatelné logické bloky. Jsou složeny z několika menších prvků nazvaných SLICE. Jejich počet se může lišit dle architektury. SLICE mají méně logických výstupů než vstupů a také vstupy a výstupy pro přenos carry bitů při konstrukci sčítaček. Switch slouží pro propojení jednotlivých CLB s globální propojovací maticí. Zároveň obsahují obvody pro připojení k nejbližším sousedním CLB.

[1] Např. fáze načtení a dekódování instrukcí



Obrázek 2.1: Architektura FPGA čipu, převzato z [10]

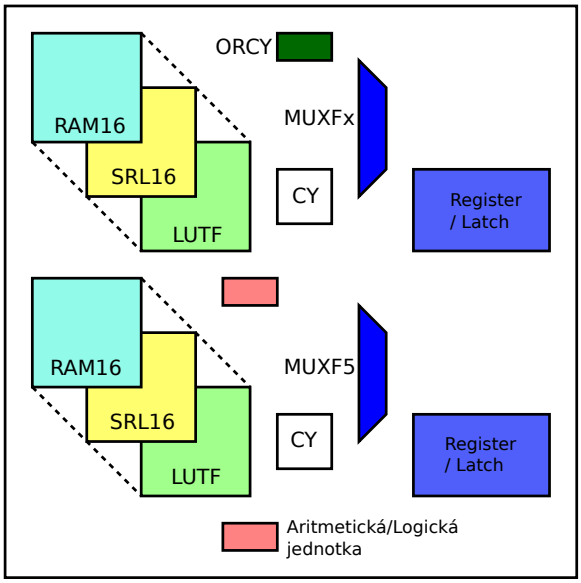


Obrázek 2.2: Struktura CLB, převzato z [10]

Struktura SLICE

SLICE obsahuje funkční generátory, které mají několik možností využití. Funkční generátor je použit např. jako Look-Up Table (dále LUT). Realizuje libovolnou funkci N logických proměnných. Tato funkce definuje chování SLICE přiřazením výstupní hodnoty na základě vstupních hodnot. Slouží tedy pro nastavení konfigurace logického obvodu, a tedy celého FPGA. Dalším využitím je použití LUT jako rychlé RAM paměti s menším obsahem $2^N - 1$ bitů. V tomto případě slouží vstupy obvodu jako adresové bity, dále musí být generátor vybaven datovým vstupem, vstupem pro povolení zápisu a vstupem pro hodinový signál

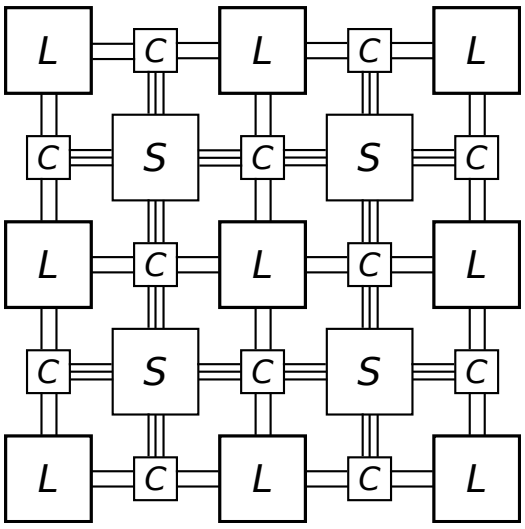
při synchronním zápisu.



Obrázek 2.3: Struktura SLICE, převzato z [10]

2.1.2 Propojovací síť

Propojovací síť FPGA je tvořena vertikálními a horizontálními vodiči, lze ji vidět na obrázku 2.4.



Obrázek 2.4: Propojovací síť FPGA, převzato z [10]

Konfigurační bloky jsou na propojovací síť připojeny pomocí C-boxů. V místě křížení vertikálních a horizontálních vodičů jsou umístěny přepínače, pomocí kterých dochází ke konfiguraci a vytvoření jednotlivých propojení v propojovací síti. Propojovací síť tvoří značnou část plochy FPGA čipu. Při vytváření hardwarové aplikace je třeba dávat pozor na

zpoždění jednotlivých propojení spolupracujících komponent. Nastavení zpoždění jednotlivých propojení lze realizovat pomocí tzv. constrains[2].

2.1.3 Vstupy a výstupy

Vstupně - výstupní bloky (na obrázku 2.1 bloky IOB's) slouží pro komunikaci aplikace uvnitř FPGA čipu s vnějším okolím. Zároveň slouží pro nahrávání konfigurace. Pro každý IO blok lze nastavit použití registru pro lepší časování a synchronizaci uvnitř čipu. Při vytváření aplikace je nutné přiřadit všem VHDL vstupním a výstupním portům konkrétní piny pouzdra čipu. Zároveň lze pomocí constrains nastavit normu pro vstupní a výstupní napětí a hladiny logických hodnot[3].

2.1.4 Zachování konfigurace čipu

Jelikož o uložení konfigurace se starají SRAM paměti, které k udržení informace potřebují připojené napájení, je třeba při připojení napájení k čipu nejprve konfiguraci nahrát. O uchování konfiguračních informací se stará externí paměť EEPROM nebo Flash[3]. Zároveň tato paměť zachová uložený bitstream v případě odpojení či přerušení napájení.

2.2 Karta COMBOv2

Karta COMBOv2 byla vytvořena pro usnadnění vývoje hardwarových aplikací. Základem karty je minimálně jeden nebo více znovuprogramovatelných FPGA čipů. Díky velkému obsazení dalšími komponentami, jako např. síťové prvky, konektory pro připojení paměťových karet a dalších periferií, má karta COMBOv2 širokou oblast použití. Karta tvoří modulární systém, který se skládá ze základní karty doplněné o karty s rozhraním a karty pro zařízení s nižšími rychlostmi. Karta obsahuje endpoint pro připojení k PCI Express sběrnici, lze ji tedy připojit do stolního počítače. Bližší informace o kartách rodiny COMBOv2 lze nalézt v [26].

Obrázek 2.5: Karta COMBOv2, převzato z [26]

Jednotlivé typy karet jsou také rozděleny podle typu FPGA čipu, kterým jsou osazeny. Karta je osazena technologií Virtex-5 od výrobce Xilinx. Čipy této rodiny obsahují různé

[2] Omezující podmínky kladené pro dosažení požadované výkonnosti.
[3] TTL,CMOS atd.

vybavení pro usnadnění práce a konkrétní požadavky kladené na vytváření akcelerovaných zařízení. Pro vývoj síťových aplikací je použita platforma označená LXT, určená pro tvorbu logických aplikací s prostředky pro pokročilou sériovou komunikaci. Obecně čipy označené xXT obsahují integrované Ethernetové MAC bloky, které umožňují zpracovat přicházející a odesílané Ethernetovské rámce. Pro seznámení se všemi typy FPGA čipů lze nahlédnout do [34].

Hlavní motivací pro vytvoření karty je použití v oblasti síťových technologií. Pro toto použití lze k základní kartě připojit modul s rozhraními pro připojení 2x10 Gb/s (karta COMBOI-10G2) nebo 4x1G b/s (karta COMBOI-1G4).

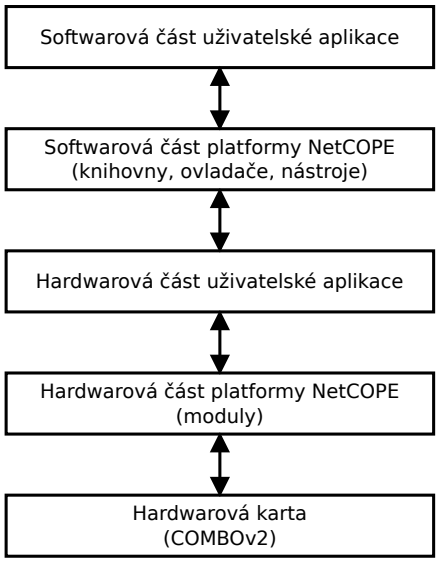
Kapitola 3

Platforma NetCOPE

Tato kapitola popisuje platformu NetCOPE, vytvořenou pro rychlejší vývoj hardwarově akcelerovaných síťových aplikací. Popisuje celkovou architekturu platformy, přibližuje jednotlivé moduly obsažené ve firmwaru a softwarové ovladače potřebné pro komunikaci s počítačem. Platforma zároveň obsahuje i softwarové ovladače a knihovny pro připojení klientské části aplikace. Kapitola čerpá informace z dokumentace projektu Liberouter [24], [23] a [9].

3.1 Architektura platformy

Platformu NetCOPE si lze představit jako složení několika vrstev, které spolupracují, přičemž každá vrstva má přesně definované rozhraní. Jednotlivé vrstvy platformy tak, jak po sobě následují, lze vidět na obrázku 3.1.



Obrázek 3.1: Vrstvy platformy NetCOPE, převzato z [24]

Hardwarová karta Obsahuje samotný FPGA čip a přídavné obvody. Jednou z podporovaných karet je karta COMBOv2 popsaná v kapitole 3.2

Hardwarová část platformy NetCOPE Jedná se o hardwarovou část platformy, která se liší dle použité karty. Pro každou kartu je tedy třeba vytvořit vlastní firmware,

lze však využít některé hardwarově nezávislé komponenty. Patří sem moduly síťového rozhraní, modul pro připojení na sběrnici a další moduly pro ovládání a konfiguraci karty.

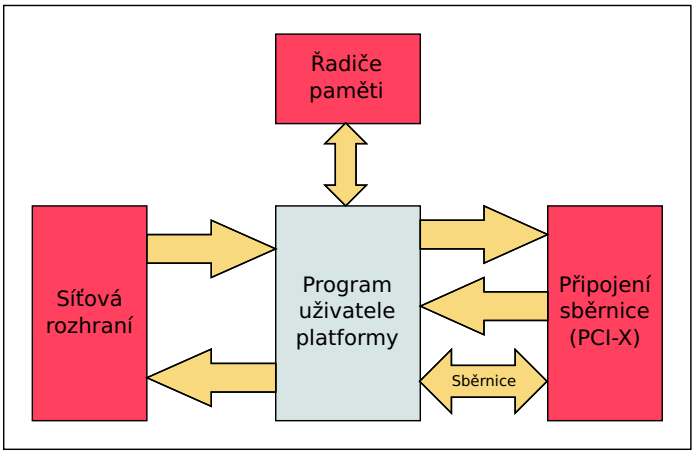
Hardwarová část uživatelské aplikace Tato vrstva obsahuje implementaci výkonově náročných částí uživatelské aplikace. Rozhraní, přes které je realizována komunikace s hardwarovou částí platformy, dosahuje abstrakce potřebné pro nezávislost na použité hardwarové kartě.

Softwarová část platformy NetCOPE Obsahuje systémové ovladače hardwarové karty a poskytuje rozhraní pro softwarovou část uživatelské aplikace. Také obsahuje pomocné nástroje pro vývoj a testování hardwarové části aplikace. Tato část je popsána blíže v kapitole 3.9

Softwarová část uživatelské aplikace Sem patří méně kritické části uživatelské aplikace. Výhodou této vrstvy je dostatečná flexibilita pro realizaci celé aplikace. Komunikace s hardwarovou částí je realizována pomocí nižší vrstvy softwarových knihoven.

3.2 Hardwarová část platformy

Hardwarová část platformy je znázorněna na obrázku 3.2.



Obrázek 3.2: Hardwarová část platformy NetCOPE, převzato z [24]

Skládá se z propojovacích sběrnic sloužících pro komunikaci mezi jednotlivými částmi platformy pomocí přesně definovaných komunikačních protokolů a modulů pro manipulaci s datovými toky, modulu zajišťujícího rozhraní mezi fyzickou vrstvou počítačové sítě a jádrem aplikace, modulu pro přímý přístup do paměti hostujícího počítače a modulu uživatelské aplikace. Dalším důležitým modulem platformy je např. generátor přesných časových razítek a modul pro identifikaci nahraného firmwaru.

Pro správné použití jednotlivých modulů a sběrnic je třeba důkladné seznámení s komunikačními protokoly na sběrnicích a rozhraními jednotlivých modulů, popsaných v následujících kapitolách.

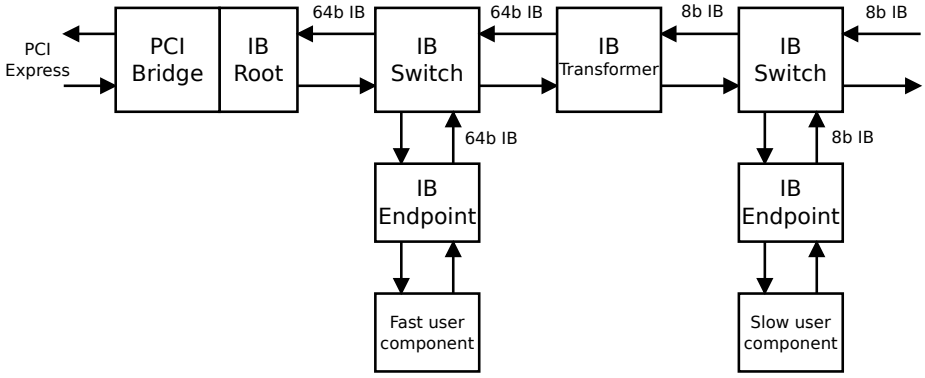
3.3 Komunikační sběrnice a protokoly

K propojení jednotlivých komponent platformy je implementováno několik druhů komunikačních sběrnic. Ty jsou využity pro propojení aplikačního jádra s moduly a připojení všech komponent ke sběrnici PCI Express pro komunikaci se softwarovou částí aplikace. Kapitola stručně seznamuje se základními principy použitých komunikačních sběrnic.

3.3.1 Interní sběrnice

Jedná se o hlavní komunikační sběrnici v rámci platformy NetCOPE. Slouží pro přenos dat a příkazů v hardwarové části aplikace. Sběrnice je ukončena komponentou PCI Express Bridge, která slouží jako převodník na sběrnici PCI Express, na kterou je karta připojena k počítači. Typicky jsou na sběrnici připojeny DRAM paměti, lze připojit i např. PowerPC procesor. Interní sběrnice má typicky bitovou šířku 8,16,32 nebo 64 bitů. Každá linka je plně duplexní[1].

Jedná se o stromovou strukturu sestavenou z kořene (root), přepínačů (switch), převodníků (transformers) a koncových uzlů (endpoint). Jak je vidět na obrázku 3.3, lze stromovou strukturu reprezentovat jako sběrnici, kde jsou přepínače použity jako odbočky z hlavní části sběrnice.



Obrázek 3.3: Ukázková architektura interní sběrnice, převzato z [24]

Komunikace po interní sběrnici je založena na paketech. Každý paket se skládá ze 128 bitové hlavičky, podle typu transakce pak mohou následovat data o velikosti 1–4096 bajtů. Maximální délka paketu je určena podle specifikace PCI Express sběrnice a maximální velikosti stránky v paměti.

Typy transakcí

Interní sběrnice rozeznává dva druhy transakcí, a to lokální a globální. Lokální transakce realizují přenos dat mezi komponentami uvnitř FPGA čipu, globální transakce přenášejí data v celém dostupném adresovém prostoru, který je přístupný přes PCI Bridge.

Dále lze dělit transakce podle operace, která je s jednotlivými pakety provedena:

- Zápisová a dokončovací - obsahují hlavičku i data

[1] Data mohou jít zároveň oběma směry

- Čtecí - obsahuje pouze hlavičku s požadavkem na čtení, dokončovací transakce je odpovědí

Přesný popis formátu jednotlivých transakcí lze nalézt v [11].

Komponenty

Sběrnice obsahuje několik komponent sloužících pro vybudování topologie sběrnice, propojení jednotlivých komponent vytvářené aplikace a připojení interní sběrnice na sběrnici PCI.

Switch Slouží pro přepínání transakcí ze vstupních portů na výstupní podobně jako přepínač v síťové topologii. K identifikaci cíle slouží cílová adresa uvedená v hlavičce paketu.

Endpoint Jedná se o koncový bod sběrnice. Sem lze připojit moduly, u kterých je třeba zajistit komunikaci po interní sběrnici.

Transformer Slouží pro spojení dvou částí interní sběrnice s různou šířkou datového kanálu, zajišťuje konverzi jednotlivých transakcí.

Root Tvoří zakončení interní sběrnice a realizuje konverzi na signály sběrnice PCI Express.

Adresový prostor

Adresový prostor interní sběrnice je adresován 32 bitovou adresou. Rozdělení tohoto prostoru je uvedeno v tabulce 3.1.

3.3.2 Paměťové rozhraní

MI32 sběrnice je jednoduché rozhraní pro připojení komponent s menší rychlostí přenosu dat. Proto toto rozhraní není tak náročné na zdroje FPGA čipu. Je vhodné pro konfiguraci datových přenosů nebo pro čtení statových a ladících informací.

Pro práci s paměťovým rozhraním je vytvořeno několik pomocných modulů:

MI Splitter Umožňuje připojit několik jednotek k jednomu paměťovému rozhraní.

EPI to MI Converter Slouží pro připojení k interní sběrnici. Zajišťuje převod signálů mezi interní sběrnicí a paměťovým rozhraním.

Pipe Slouží pro vylepšení časování sběrnice.

MI8 to MI32 Transformer Slouží ke konverzi 8 bitového MI rozhraní na 32 bitové.

Sběrnice použita v NetCOPE je označena jako MI32, protože šířka datové i adresové cesty je 32 bitů.

Využití	Od	Do	Velikost
ID komponenta	0x00000000	0x000003FF	1 KiB
I2C / MDIO	0x00000400	0x000007FF	1 KiB
Nevyužito	0x00000800	0x00003FFF	14 KiB
Síťový modul	0x00004000	0x00007FFF	16 KiB
Timestamp jednotka	0x00008000	0x0000BFFF	16 KiB
DMA modul - kontrolní prostor	0x0000C000	0x0000FFFF	16 KiB
Rychlé rozhraní 1	0x00010000	0x0001FFFF	64 KiB
Rychlé rozhraní 2	0x00020000	0x0002FFFF	64 KiB
Pomalé rozhraní 1	0x00030000	0x0003FFFF	64 KiB
Pomalé rozhraní 2	0x00040000	0x0004FFFF	64 KiB
Pomalé rozhraní 3	0x00050000	0x0005FFFF	64 KiB
Pomalé rozhraní 4	0x00060000	0x0006FFFF	64 KiB
Nevyužito	0x00070000	0x0007FFFF	64 KiB
MI32 sběrnice	0x00080000	0x01FFFFFF	31,5 MiB
DMA modul - datový a IRQ prostor	0x02000000	0x022FFFFF	3 MiB
IB sběrnice	0x02300000	0x03FFFFFF	29 MiB
Nevyužito	0x04000000	0xFFFFFFEF	3,937 GiB
PCI Express Bridge	0xFFFFFFF0	0xFFFFFFFF	16 B

Tabulka 3.1: Adresový prostor interní sběrnice

3.3.3 FrameLink a Multilink sběrnice

FrameLink sběrnice

Jedná se o dedikovanou univerzální sběrnici pro připojení bufferů síťového modulu k DMA modulu. Je využíván aplikacemi určenými pro klasifikaci dat.

Komunikační protokol sběrnice vychází z protokolu LocalLink [32]. Komunikace přes sběrnici se odehrává pomocí paketů definovaných v NetCOPE. Paket se skládá z hlavičky, dat a zápatí, přičemž každá část může být vynechána. Zároveň je možno připojovat další data, obsahující doplňující informace či statistiky k právě zpracovávanému paketu pomocí tzv. *parts*, které jsou přeneseny pomocí datových rámců připojených za základní částí paketu. Rámec může obsahovat i více částí. Této možnosti využívá modul PACODAG popsaný v podkapitole 3.4.

DATA Datová sběrnice pro přenos dat. Nejčastěji používané šířky sběrnice jsou 16,32,64 a 128 bitů.

DREM Signál slouží příjemci k detekci posledního validního bajtu v aktuální přenášené části. Pozice je binárně zakódována v signálu o šířce $log_2(n)$.

SOF_N Indikuje hodinový takt, ve kterém je na sběrnici první rámec dat

EOF_N Indikuje hodinový takt, ve kterém je na sběrnici poslední vyslaný rámec dat

SOP_N Indikuje hodinový takt, ve kterém je na sběrnici první část dat

EOP_N Indikuje hodinový takt, ve kterém je na sběrnici poslední část dat

Název signálu	Bitová šířka	Směr
DATA	8-64	Zdroj → Cíl
DREM	1-3	Zdroj → Cíl
SOF_N	1	Zdroj → Cíl
EOF_N	1	Zdroj → Cíl
SOP_N	1	Zdroj → Cíl
EOP_N	1	Zdroj → Cíl
SRC_RDY_N	1	Zdroj → Cíl
DST_RDY_N	1	Cíl → Zdroj

Tabulka 3.2: Signály FrameLink sběrnice

SRC_RDY_N Signalizuje, zda je zdroj připraven na odesílání dat

DST_RDY_N Signalizuje, zda je cíl připraven na příjem dat

MultiLink sběrnice

MultiLink je odvozen od sběrnice FrameLink. MultiLink nahrazuje propojení několika FrameLink, protože při takovém propojení je možné ušetřit zdroje, které by při použití několika FrameLink sběrnic zůstaly přiřazené. Slouží hlavně pro přenos dat přes DMA moduly, které umožňují data rozdělit na několik procesorových jader. Rozhraní MultiLink sběrnice se podobá rozhraní FrameLink sběrnice.

3.4 Modul Pacodag

Komponenta Pacodag (**PA**cket **CO**ntrol **DA**ta **G**enerator) slouží k vytváření kontrolních dat pro každý paket, který projde přes IBUF buffer, blíže popsaný v 3.6. Kontrolní data jsou připojena ke každému paketu podobně jako FrameLink hlavička.

Komunikační rozhraní sloužící pro komunikaci s IBUF je vytvořeno signály, které jsou jednotlivě vypsány v tabulce 3.3.

Signály nazvané CTRL_DATA, CTRL_DREM, CTRL_SRC_RDY_N, CTRL_SOP_N, CTRL_EOP_N a CTRL_DST_RDY_N jsou zjednodušené signály FrameLink protokolu. Odesílatelem dat je modul PACODAG, přičemž jsou data přijata do vstupního bufferu síťového modulu IBUF. Do zpracovávaného rámce jsou v IBUF bufferu přidány informace vytvořené modulem PACODAG. Význam ostatních signálů lze, stejně jako časování modulu, nalézt v [23].

3.5 Modul uživatelské aplikace

Modul aplikačního jádra je v hardwarové části platformy umístěn mezi modulem DMA a síťovým modulem. Vytvářená uživatelská aplikace je umístěna na rozhraní FrameLink nebo MultiLink. V tomto modulu se provádí vlastní činnost hardwarové části aplikace a případně jsou získaná data přenesena do paměti počítače pro softwarovou část aplikace. Zároveň je možné úplně vynechat hardwarovou část a pouze předávat data softwarové části aplikace. Popis softwarové části platformy NetCOPE lze nalézt v podkapitole 3.9.

Název signálu	Bitová šířka [b]	Kontrolováno
CTRL_CLK	1	IBUF
CTRL_DATA	128	PACODAG
CTRL_REM	4	PACODAG
CTRL_SRC_RDY_N	1	PACODAG
CTRL_SOP_N	1	PACODAG
CTRL_EOP_N	1	PACODAG
CTRL_DST_RDY_N	1	IBUF
SOP	1	IBUF
CTRL_RDY	1	PACODAG
STAT_PAYLOAD_LEN	16	IBUF
STAT_FRAME_ERROR	1	IBUF
STAT_CRC_CHECK_FAILED	1	IBUF
STAT_MAC_CHECK_FAILED	1	IBUF
STAT_LEN_BELOW_MIN	1	IBUF
STAT_LEN_OVER_MTU	1	IBUF
STAT_DV	1	IBUF

Tabulka 3.3: Rozhraní komponenty PACODAG

Modul uživatelské aplikace má k dispozici dva paměťové moduly. První modul je adresován od adresy $0x02300000$ o velikosti 29 MiB a slouží pro adresování z interní sběrnice platformy. Druhý paměťový blok je adresován přes rozhraní MI32, začíná na adrese $0x00080000$ a má velikost 31.5MiB.

3.6 Síťový modul

Síťový modul zajišťuje komunikaci mezi hardwarovým jádrem uživatelské aplikace a síťovým rozhraním samotné karty. Platforma NetCOPE využívá síťových bloků XGMII,GMII a EMAC integrovaných na čip FPGA.

Obsahuje buffery IBUF a OBUF pro plynulé zpracování jednotlivých rámců a zvýšenou spolehlivost. Buffery také zabraňují zahazování rámců při zahlcení sítě. Síťový modul provádí konverzi z rozhraní používaných síťovými bloky na rozhraní používané v rámci platformy NetCOPE. Síťový modul znázorňuje obrázek 3.4.

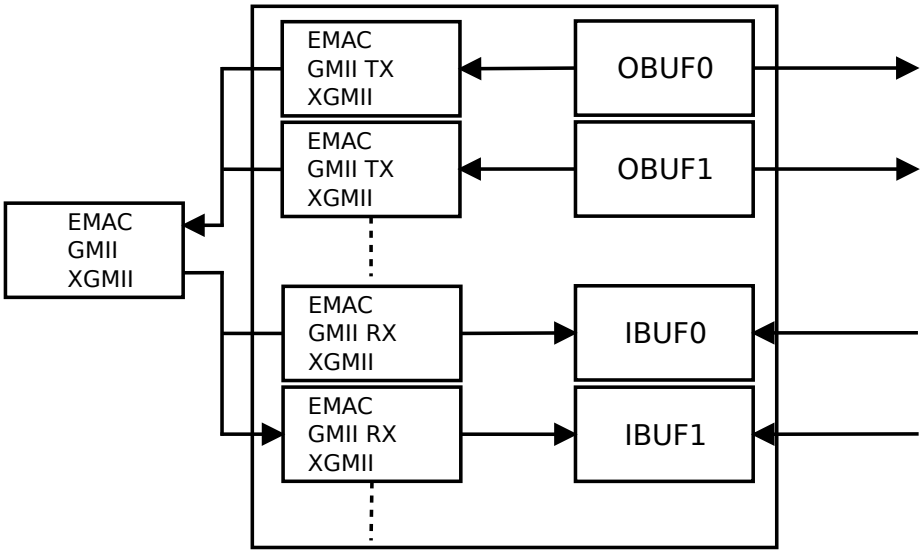
Síťový modul vždy obsahuje také rozhraní pro připojení na MI32 sběrnici, připojení na jednotku generující kontrolní data pro Ethernetové rámce a jednotku pro vzorkování. Podle počtu a typu kanálů také obsahuje určitý počet vstupních a výstupních rozhraní.

EMAC Ethernet Media Access Control, věstavěný modul schopný zpracovávat Ethernetovou komunikaci s rychlostmi 10/100/1000 Mb/s.

GMII Gigabit Media Independent Interface je modul zpracovávající rychlost komunikace 1 Gb/s.

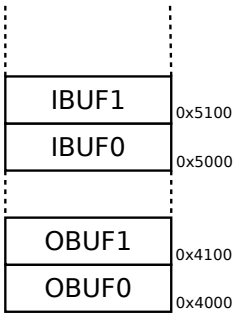
XGMII 10 Gigabit Media Independent Interface dokáže zpracovat 10 Gb/s.

OBUF,IBUF Jedná se o komponenty vstupní a výstupní buffer.



Obrázek 3.4: Síťový modul platformy NetCOPE, převzato z [24]

Adresový prostor síťového modulu je umístěn od adresy $0x4000$ s velikostí $16kB$. Na adrese $0x4000$ se nachází adresa bufferu OBUF0, přičemž vždy další buffer má adresu o $0x100$ vyšší. Situaci znázorňuje obrázek 3.5.



Obrázek 3.5: Adresový prostor síťového modulu

K jednotlivým registrům každého bufferu lze přistoupit pomocí bázové adresy bufferu dle výše uvedené tabulky a offsetu konkrétního registru daného bufferu. Registry mají v rámci vstupních bufferů identické offsety, stejně jako v rámci výstupních bufferů. Adresové prostory obou typů bufferů znázorňují tabulky 3.4 a 3.5.

3.7 Modul pro DMA

Modul pro přímý přístup do paměti slouží pro komunikaci s RAM pamětí hostujícího počítače. Je připojen na interní sběrnici a řídí přenosy dat do RAM paměti počítače. Konfigurační registry jsou dostupné z lokální sběrnice MI32. Modul obsahuje také FrameLink rozhraní pro vysílání a MultiLink rozhraní pro příjem. Adresový prostor DMA řadičů lze vidět v tabulce 3.6.

Další řadiče RX a TX mají adresu vždy o $0x40$ větší. Registry pro přenos dat pomocí

Význam registru	Offset adresy
Celkový počet rámců - LOW	$0x00$
Počet odeslaných rámců - LOW	$0x04$
Počet zahozených rámců - LOW	$0x08$
Celkový počet rámců - HIGH	$0x10$
Počet odeslaných ramců - HIGH	$0x14$
Počet zahozených rámců - HIGH	$0x18$
Povolovací registr	$0x20$
Registr MAC adresy - LOW	$0x24$
Registr MAC adresy - HIGH	$0x28$
Příkazový registr	$0x2C$
Stavový registr	$0x30$

Tabulka 3.4: Offsety registrů OBUF

Význam registru	Offset adresy
Celkový počet přijatých rámců - LOW	$0x00$
Počet správně přijatých rámců - LOW	$0x04$
Počet zahozených rámců - LOW	$0x08$
Počet zahozených rámců z důvodu zaplnění bufferu - LOW	$0x0C$
Celkový počet přijatých rámců - HIGH	$0x10$
Počet správně přijatých rámců - HIGH	$0x14$
Počet zahozených rámců - HIGH	$0x18$
Počet zahozených rámců z důvodu zaplnění bufferu - HIGH	$0x1C$
Povolovací registr	$0x20$
Registr pro nastavení maskování chyb	$0x24$
Stavový registr	$0x28$
Registr pro příkazy	$0x2C$
Minimální povolená velikost rámce	$0x30$
MTU rámce	$0x34$
Režim registrace MAC adresy	$0x38$
Paměť volných MAC adres	$0x80$

Tabulka 3.5: Offsety registrů IBUF

Význam registru	Offset adresy
RX DMA řadič 0	$0xC000$
TX DMA řadič 0	$0xC040$
RX DMA řadič 1	$0xC080$
TX DMA řadič 1	$0xC0C0$

Tabulka 3.6: Konfigurační registry DMA modulu

Význam registru	Offset adresy
RX Buffer 0	$0x02000000$
RX Buffer 1	$0x02002000$
⋮	⋮
TX Buffer 0	$0x02100000$
TX Buffer 1	$0x02102000$
⋮	⋮
Descriptor Manager	$0x02200000$
RX stavový registr	$0x02280000$
TX stavový registr	$0x02280008$

Tabulka 3.7: Datové registry DMA modulu

DMA mají adresový prostor vyhrazený na interní sběrnici. Zde se také nachází stavové registry pro oba směry přenosu.

3.8 Modul časových razítek

Jednotka pro generování časových razítek je využita např. v aplikacích monitorujících síťový provoz. Pro správnou funkčnost je třeba možnost přijímat GPS signál přes kartu s GPS modulem. Modul je pak synchronizován s GPS systémem. Jinak generuje vlastní, méně přesnou časovou značku. Výstup modulu se skládá ze tří signálů.

TS 64 bitový timestamp. Horních 32 bitů obsahuje počet sekund od 1. 1. 1970, spodních 32 bitů reprezentují části sekund

TS_DV 1 bit označující validní data TS

TS_CLK Hodinový signál

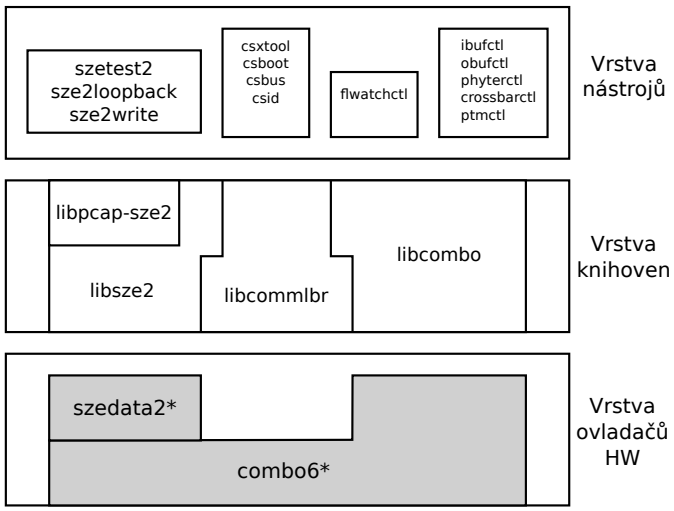
3.9 Softwarová část platformy

Softwarová část NetCOPE zajišťuje komunikaci hardwarové části platformy s operačním systémem hostujícího počítače pomocí definovaného rozhraní. Softwarová část se skládá ze systémových ovladačů, potřebných pro spolupráci hardwarové karty s operačním systémem, dále knihovny, které slouží pro ovládání hardwarové karty ze softwarové části uživatelské aplikace a nástroje pro konfiguraci komponent hardwarové karty a pro testování a ladění.

Jednotlivé vrstvy jsou uvedeny na obrázku 3.6.

Vrstva ovladačů obsahuje ovladače combo6* a szedata2*. Ovladač combo6* slouží pro práci s kartami rodiny COMBO a zabezpečuje vstup, výstup a navázání komunikace mezi kartou a hostujícím počítačem. Szedata2* slouží pro využívání agregovaných DMA přenosů. Podrobnější informace o ovladačích lze získat např. [6], kde je funkčnosti ovladače věnována celá kapitola.

Vrstva knihoven tvoří nadstavbu nad vrstvou ovladačů a vytvářejí abstrakci pro ovládání hardwarové karty. Knihovna libcommlbr obsahuje nástroje pro ladění, správu verzí firmwaru a rutiny vykonávající často se opakující činnosti. Libcombo knihovna slouží pro bootování



Obrázek 3.6: Softwarové vrstvy NetCOPE platformy, převzato z [24]

a inicializaci firmwaru a vstupně-výstupní operace. Libsze2 poskytuje rozhraní pro rychlé DMA přenosy oběma směry. Vrstva obsahující nástroje vytváří uživatelské rozhraní pro bootování, testování, konfiguraci a monitorování hardwarové karty.

Kapitola 4

Firewally

Tato kapitola se zabývá popisem síťového zařízení zvaného firewall. Slouží k řízení a zabezpečení síťového provozu mezi dvěmi různými sítěmi. Firewall se řídí tzv. pravidly, která definují povolený nebo zakázaný datový tok[1] mezi sítěmi. Jedny z prvních firewallů používaly pro identifikaci datového toku zdrojovou a cílovou IP adresu a zdrojový a cílový port. S postupem času však tyto vlastnosti datového toku přestaly být dostačující, proto novější firewally sledují alespoň informace o stavu datového toku, využívají znalostí kontrolovaných protokolů, nejpokročilejší firewally pak mohou implementovat prvky IDS[19].

Firewallem může být buď specializované hardwarové zařízení, nebo specializovaný software běžící na klasickém počítači či serveru.

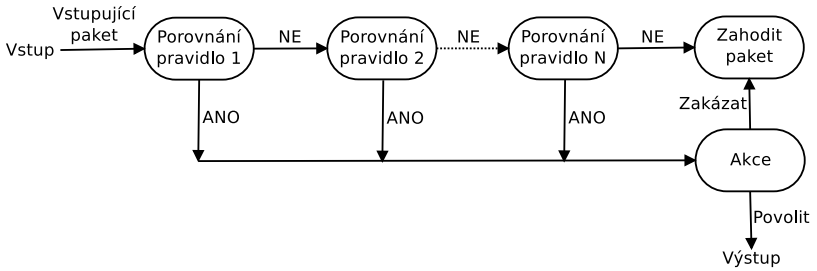
Kapitola o firewallech vychází z knih [22] a [21].

4.1 Filtrovací pravidla

Chování firewallu je řízeno tzv. filtrovacími pravidly. Síťový provoz je kontrolován filtrovacími pravidly a podle výsledku porovnání data propustí nebo zablokují. Jednoduchý příklad, jak mohou vypadat tato pravidla, je uveden zde:

```
<number> <action> <protocol> from <src IP> to <dest IP>
  [src-port <srcPort>] [dst-port <dstPort>] [<flags>] [<options>]
```

Vyhodnocování filtrovacích pravidel probíhá v pořadí atributu `<number>` podle obrázku 4.1.



Obrázek 4.1: Princip vyhodnocování pravidel

[1] Posloupnost paketů se stejnými vlastnostmi a zároveň procházejí stejnými uzly v síti

Pro názornost můžeme uvažovat tato pravidla:

```
60 deny IP any any
10 permit TCP from 147.229.0.0 to any dst-port 80
50 deny ICMP from 147.229.1.15 to any
20 permit UDP from 147.229.0.0 to any dst-port 53
```

Při příchozím paketu je nejprve ověřeno, jestli tento paket vyhovuje pravidlu s atributem `<number>` 10. Pokud ano, je pravidlo vyhodnoceno. V tomto případě je tedy povoleno TCP spojení ze sítě `147.229.0.0` s libovolnou adresou destinace a cílovým portem 80 (protokol HTTP). Pokud paket vyhovuje zadaným kritériím, je umístěn na výstupní rozhraní. Pokud ne, stejným způsobem se vyhodnocuje pravidlo s `<number>` 20. Toto pravidlo povoluje UDP komunikaci ze sítě `147.229.0.0` s libovolnou adresou destinace a cílovým portem 53 (protokol DNS).

Pravidlo s `<number>` 50 zakazuje ICMP komunikaci z uzlu s IP adresou `147.229.1.15`, čili pokud dorazí paket vyslaný např. programem `ping` z tohoto počítače, bude zahozen. Poslední pravidlo s parametrem `<number>` 60 udává, že všechny pakety nevyhovující předchozím pravidlům, budou zahozeny.

Pravidla firewallu nastavuje správce počítačové sítě manuálně. Pravidla se dělí na dva druhy podle doby platnosti. Statická pravidla platí stále, oproti tomu dynamická pravidla mění platnost podle zadaných podmínek. Dále lze dělit filtrování dat podle výchozího přístupu. Inkluzivní přístup povoluje pouze datové toky, které vyhovují pravidlům, ostatní blokuje. Je bezpečnější než exkluzivní přístup. Exkluzivní přístup propouští všechny datové toky kromě těch, které pravidla zakazují.

4.2 Umístění firewallu

Jak již bylo uvedeno výše, firewall slouží pro filtraci provozu mezi dvěma sítěmi. Nejrozšířenější druhy použití jsou dva, a to firewall zapojený jako síťový most a firewall s demilitarizovanou zónou.

4.2.1 Režim síťového mostu

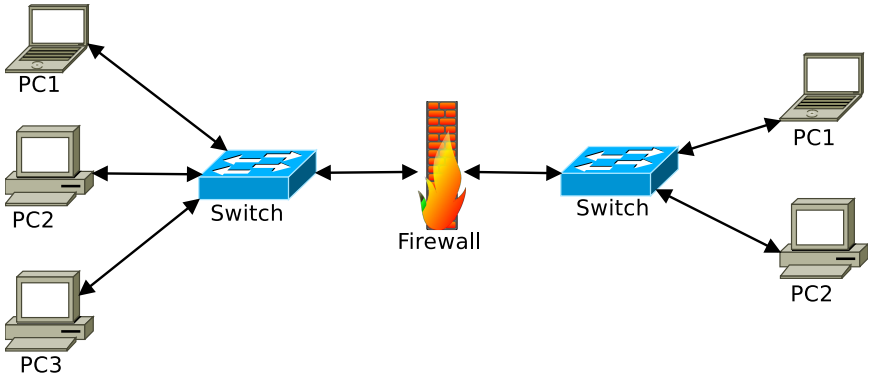
Firewall je z pohledu sítě plně transparentní, příchozí data na vstupním rozhraní zkontroluje podle filtrovacích pravidel a předává je na výstupní rozhraní. Topologii použití jako síťový most znázorňuje obrázek 4.2.

4.2.2 Demilirarizovaná zóna

Zapojení s demilitarizovanou zónou se používá hlavně pro větší sítě, přičemž síť rozdělí do tří částí s různou politikou.

Bezpečná část Je na obrázku znázorněna zeleně, veškerá zdejší komunikace je oddělena od zbytku světa.

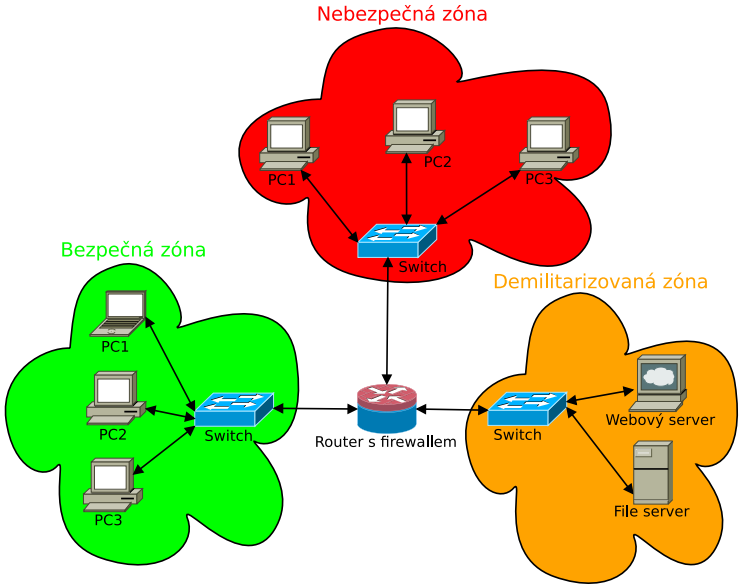
Demilitarizovaná zóna Na obrázku oranžová. Zde jsou umístěny servery, které je třeba mít dostupné z vnitřní i z vnější části sítě (DNS servery, webové servery, e-mailové servery atd.).



Obrázek 4.2: Firewall v režimu síťového mostu

Vnější síť Červená. Z této části sítě není možná komunikace se síťovými uzly v bezpečné zóně.

Schéma topologie sítě s DMZ je znázorněno na obrázku:



Obrázek 4.3: Firewall při vytvoření demilitarizované zóny

4.3 Paketové filtry

Postupný vývoj firewallů vytvořil jejich rozdělení do skupin podle principu filtrování procházejícího datového toku. Firewally jsou děleny podle hloubky kontroly datových toků na následující skupiny.

Nejstarší, ale také nejjednodušší princip filtrování provozu je založen na přesné identifikaci zdroje a cíle provozu v rámci pravidel, konkrétně z jaké IP adresy a portu může paket přicházet a na jakou IP adresu a port může být doručen. Klasifikace paketů je tedy prováděna na síťové a transportní vrstvě ISO/OSI modelu. K filtrování lze také použít

typ protokolu. Výhodou paketových filtrů je jejich jednoduchost implementace a vysoká propustnost paketů.

4.3.1 Filtrování protokolů

Filtrování protokolů probíhá pomocí položky `Protocol` v hlavičce IP datagramu. Tato položka umožňuje od sebe odlišit následující protokoly, které jsou zapouzdřeny uvnitř datagramu. Čísla jednotlivých protokolů jsou definovány v [18]. Bohužel tato položka je příliš obecná, protože umožňuje odlišit pouze výše uvedené protokoly. Z toho důvodu je třeba všechny protokoly mít otevřené.

4.3.2 Filtrování IP adres

Pomocí kontroly IP adres lze filtrovat a omezit provoz na vybrané koncové počítače nebo sítě, případně zakázat provoz z vybraných koncových počítačů či z celých sítí. Při použití exkluzivního přístupu zakážeme ty IP adresy počítačů, se kterými chceme zablokovat komunikaci. To však nepřináší řešení, jelikož dopředu není známo, z jaké IP adresy bude veden případný útok. Silnou ochranu poskytuje použití inkluzivního přístupu, kdy lze omezit komunikaci pouze s těmi IP adresami, o kterých víme, že jsou bezpečné. Takto lze povolit komunikaci např. zaměstnancům pracujícím z domu, jiným pobočkám atd.

Útočník může zjistit IP adresu pomocí přímého směrování. Útočník do paketu umístí povolenou adresu a zachytí zpětný provoz směrovaný na jeho počítač. Proto je třeba paketový filtr nastavit tak, aby nepropouštěl dále pakety s přímým směrováním. Kvalitnější paketové filtry umožňují nadefinovat přístup k jednotlivým protokolů. Lze tak omezit přístup na port 23 (Telnet) pouze na vnitřní síť, a přístup např. na porty 80 (HTTP) a 443 (HTTPS) povolit i do vnější sítě. Útočník může filtr obejít, pokud v paketu pozmění IP adresu na adresu, kterou firewall povoluje. Musí však tuto adresu znát. Tuto metodu lze použít pouze pokud není nutná zpáteční cesta (útoky Dos nebo pokud je adresa protokolu uvedena v datové části).

4.3.3 Porty

Porty jsou nejčastěji používanou vlastností datového toku při jeho filtrování. Porty umožňují nejpřesnější popis, k čemu daný paket slouží. Běžně se tedy filtrování portů označuje jako filtrování protokolů. Podobně jako u filtrování adresy lze i zde povolit všechny porty a vybrat zakázané, nebo všechny porty zakázat a poté vybrat pouze povolené porty. Seznam používaných portů pro jednotlivé služby lze nalézt např. na [8].

4.3.4 Fragmentace

Protokol IP podporuje dvě metody, které jsou sice zastaralé, ale útočníci je často využívají. Jedná se o již dříve zmiňované přímé směrování a fragmentaci. Proto je vhodné pakety s přímým směrováním či fragmentací bez výjimky zahazovat.

4.3.5 Nedostatky bezstavových paketových filtrů

Bezstavové filtry trpí dvěma nedostatky, které snižují jejich účinnost.

Neuchovávání stavu připojení

Bezstavové filtry provádějí rozhodnutí pro každý paket, a to buď povolit, nebo zakázat. Nelze tedy určit, zda se mají zahazovat fragmenty, protože fragmenty neobsahují informaci o portu služby. Problém fragmentace spočívá v tom, že informace užitečné pro filtrování se nacházejí pouze v nultém fragmentu (jedná se o informace o portech). Fragmenty 1 a výše tyto informace neobsahují. Firewall tedy fragmenty 1 až N předá dále s tím, že pokud byl 0. fragment ztracen, jsou všechny další fragmenty bezcenné. Některé závadné verze protokolu TCP/IP však i paket bez 0. fragmentu dokáží sestavit, proto tedy stačí útočníkovi upravit odesílání fragmentů tak, aby první fragment měl pořadové číslo 1. Firewall pak bez zkoumání stavu takový rámec pustí dále.

Bezstavový filtr také nerozezná zpětné připojení na soket, pokud je TCP spojení vytvářeno z vnitřní sítě. Jednodušší filtry toto řeší tak, že povolují všechny porty od čísla 1024 pro TCP spojení. V tomto rozsahu se nacházejí dynamické porty pro komunikaci. Tento rozsah se využívá kvůli zabránění vložení paketů z jiného spojení, které proběhlo dříve na stejném portu.

Útočník může této skutečnosti využít podstrčením trojského koně, který naslouchá na jednom z otevřených TCP portů. Útočník se pak připojí pomocí TCP spojení na daný port a získá přístup k napadenému počítači.

Nemožnost kontrolovat datovou část paketu

Paketové filtry provádějí rozhodování pouze na základě informací obsažených v hlavičce paketu. Nezkoumají tedy datovou část paketu. Přitom v ní se mohou nacházet nebezpečná data úmyslně podstrčená útočníkem. Příkladem může být paket obsahující HTTP protokol, který může obsahovat trojského koně vloženého do ovládacích prvků ActiveX. ActiveX má totiž přístup k celému PC, není vykonáván v sandboxu[2], podobně jako JavaScript.

4.4 Aplikační filtry

Aplikační filtry nebo-li proxy firewally či servery byly vytvořeny o něco později než paketové filtry. Slouží pro úplné oddělení dvou počítačových sítí, filtrování je realizováno na aplikační vrstvě ISO/OSI modelu. Server v tomto případě vidí v položce zdrojová adresa adresu aplikačního firewallu, takže si nemůže udělat představu o uspořádání sítě.

Výhodou těchto filtrů je kompletní náhled na komunikaci vyplývající z kontroly až na aplikační úrovni. Lze tedy vidět celý obsah komunikace včetně komunikace na nižších vrstvách. Zároveň zvyšují bezpečnost tak, že neumožňují přenášet data přes porty, které jsou určené k jinému druhu komunikace. Nevýhodu těchto filtrů je vysoká náročnost na výpočetní výkon a vysoká latence komunikace, z čehož se odvíjí nižší propustnost firewallu. Způsobuje to fakt, že takový firewall musí rozumět všem vyskytujícím se protokolům, což je výpočetně velice náročné. Bližší informace o tomto typu firewallů lze nalézt v [21].

4.5 Stavové filtry

Stavový firewall kontroluje v paketu především informace na 4. a nižších vrstvách. Pokud kontrolovaný paket vyhovuje pravidlům, je propuštěn a přidán záznam do stavové tabulky.

[2] http://cs.wikipedia.org/wiki/Sandbox

Další pakety téže komunikace mohou firewallem procházet bez podrobnějšího zkoumání, protože se shodují s položkou ve stavové tabulce.

Nejprve je třeba vysvětlit pojem *stav* z hlediska stavových firewallů. Stavem se rozumí aktuální podoba dané síťové komunikace. Přesná definice se může měnit podle konkrétní aplikace a podle použitého komunikačního protokolu. Pro potřeby stavového filtrování je třeba co nejpřesněji identifikovat komunikační relaci. K identifikaci relace lze použít zdrojovou a cílovou IP adresu, zdrojový a cílový port, typ protokolu, příznaky, pořadové a potvrzované číslo a další. Údaje ve stavové tabulce musí být co nejpřesnější, aby útočník nemohl zkonstruovat zdánlivě korektní přenos dat.

V následujících podkapitolách je uvedeno chování pro často se vyskytující protokoly.

4.5.1 TCP

Na obrázku 4.4 je znázorněno zaslání požadavku z vnitřní sítě na webový server. Při příchodu požadavku do stavového firewallu je vytvořena nová položka tabulky. Bude obsahovat informace identifikující navazované spojení, v tomto případě TCP relaci: zdrojová a cílová IP adresa, zdrojový a cílový port.

Stavový firewall tedy umí, kromě informací v hlavičce, filtrovat také pomocí kontextu paketu v komunikační relaci, tedy dokáže odlišit pakety sloužící pro navázání spojení od paketů, které realizují již navázanou komunikaci. To umožňuje povolit navazování spojení pouze z vnitřní sítě ven a opačným směrem navazování zakázat. Pakety již navázaných spojení mohou být přenášeny oběma směry. Pakety jsou označeny následujícím způsobem:

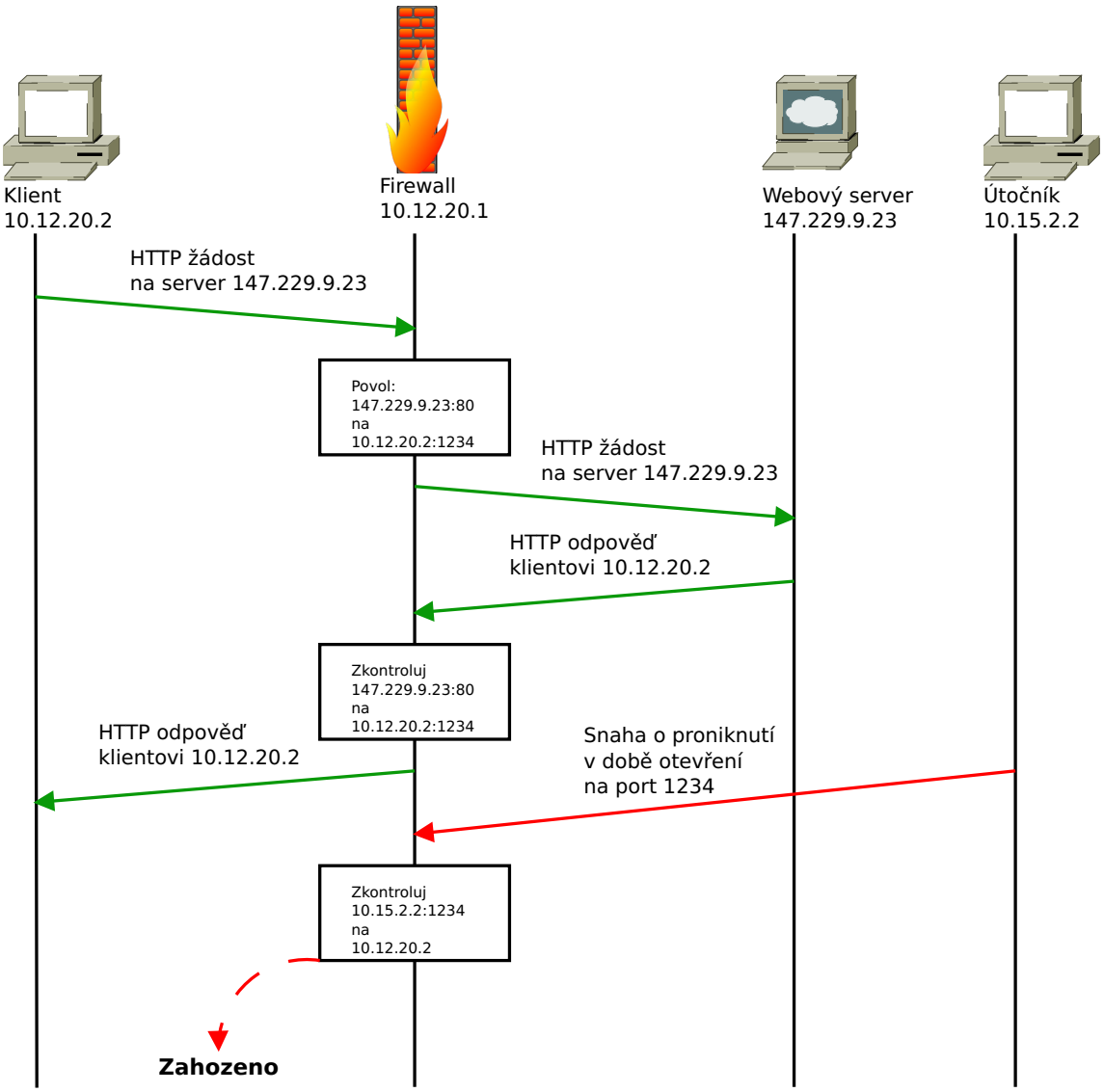
NEW - paket zahajující novou komunikaci

ESTABLISHED, RELATED - paket patří do již navázaného spojení nebo s ním nějak souvisí

INVALID - paket nepatří do žádného navázaného spojení nebo se jej nepodařilo identifikovat

Pro navázání spojení v TCP relaci se používá třícestný handshake. Zahajovací paket obsahuje nastavený příznak `SYN`. Opačná strana v případě dostupnosti požadované služby odpovídá paketem s nastavenými příznaky SYN a ACK, jehož příjem je opět potvrzen paketem s nastaveným příznakem ACK. Pro navázání spojení tedy stačí povolit pouze pakety obsahující nastavený příznak SYN, přičemž při přijetí takového paketu je vytvořen nový záznam o komunikaci do tabulky a zároveň je interně vygenerováno pravidlo pro paket obsahující odpověď druhé strany na SYN, tedy nastavené SYN a ACK příznaky. Pokud tento paket skutečně firewallem projde, je opět aktualizován stav relace v tabulce a pravidlo obsahující flagy SYN a ACK je nahrazeno pravidlem s flagy ACK. Samozřejmostí těchto pravidel je dodržení adres a portů příjemce a odesílatele pro každý paket. Při průchodu paketu s příznakem ACK je spojení považováno za navázané a vygenerují se pravidla tak, aby obě strany mohly komunikovat. Pravidlo povolující paket s ACK příznakem je odstraněno. Spojení je označeno jako `ESTABLISHED`.

Podobným způsobem lze zpracovat i ukončení TCP relace pomocí čtyřcestného handshake, kdy se po průchodu příslušných paketů odstraní záznam ze stavové tabulky. Spousta aplikací zasílá `keepalive` pakety v době nečinnosti komunikace, aby firewall neodstranil jejich záznam ze stavové tabulky. Ve stavové tabulce je také sledována doba posledního průchozího paketu dané komunikace. Při vypršení určitého časového intervalu bez aktualizace časového razítka dojde k vymazání záznamu ze stavové tabulky. Časový limit se liší



Obrázek 4.4: Příklad navázání TCP spojení a následný pokus o útok

podle toho, jestli se jedná o navázání spojení nebo o již ustanovené spojení. Při ustanovení spojení je trvanlivost záznamu maximálně minuta, aby nedošlo k přeplnění stavové tabulky např. útokem SYN-flood[3]. Při ustanoveném spojení je interval delší (až 1 hodinu), protože se předpokládá korektní a později správně ukončené spojení.

Přesné informace o činnosti TCP protokolu lze nalézt v [16].

4.5.2 UDP

Sledování stavu protokolu UDP[14] je složitější než u TCP, protože se jedná o nespojovaný protokol. Takový protokol principielně žádný stav nemá, proto je třeba jej zpracovávat pomocí pseudo-statového řízení. Pro identifikaci UDP komunikační relace lze použít pouze obě IP adresy a oba porty. Dočasná čísla portů jsou volena nahodile a pro každé spojení ze stejné

[3] http://cs.wikipedia.org/wiki/SYN_flood

IP adresy jsou různé, proto lze přesněji identifikovat jednotlivé datové toky. Při příchodu prvního UDP paketu je porovnán s pravidly a pokud vyhovuje, je ihned záznam zařazen do stavové tabulky. Jelikož protokol UDP nemá mechanismus pro ukončení spojení, dojde k odstranění záznamu ze stavové tabulky pouze při vypršení životnosti záznamu. Protokol UDP nemá žádné bezpečnostní ani opravné mechanismy, opírá se při opravě o protokol ICMP.

4.5.3 ICMP

Stejně tak jako protokol UDP, není ani ICMP stavovým protokolem. Navíc sledování ICMP komunikace je složitější zejména kvůli převážně jednosměrné komunikaci. Často se používá k vracení chybových zpráv při problémech jiných protokolů. Druhým typem ICMP komunikace je komunikace založená na principu dotaz-odpověď (např. příkaz `ping`). V tomto případě není sledování komunikace problém. Namísto sledování zdrojové a cílové IP adresy je třeba ICMP komunikaci sledovat pomocí typu zprávy a odpovědi. Opět je třeba udržovat informace v tabulce pouze určitou dobu, protože ICMP také nemá žádný mechanismus pro ukončení spojení. Informace o protokolu ICMP lze nalézt v [15].

4.5.4 FTP

Protokol FTP slouží pro přenos souborů mezi různými systémy. FTP protokol se však chová jinak než ostatní protokoly využívající TCP protokol. Při komunikaci navazuje totiž dvě spojení, která je třeba sledovat. Firewall tedy musí v případě potřeby propouštět obě spojení, proto je třeba, aby měl informace i o protokolech aplikační vrstvy s nestandartním chováním.

V prvním kroku dochází k navázání řídícího kanálu, což není problémem, protože dochází k navázání TCP komunikace. K problému dochází při vytváření datového kanálu opačným směrem než vytváření řídícího kanálu. Pokud tedy stavový firewall vidí vytváření řídícího kanálu TCP komunikace (standartně port 21), musí předpokládat navázání datové komunikace v opačném směru (standartně port 20). Firewall proto povolí komunikaci standartně na portu 20 z IP adresy cílové stanice použité při navazování řídícího kanálu.

Firewall musí ale také vědět o použitém čísle portu pro datový kanál. Tato informace je uvedena v příkazu PORT protokolu FTP. Musí provádět tedy inspekci na aplikační vrstvě. Chování FTP protokolu je definováno v [17].

4.5.5 Multimediální protokoly

Multimediální protokoly procházejí přes firewall podobně jako FTP protokol, v rámci jedné relace je však navazováno ještě více spojení. Takovým protokolem je např. RTSP[20] nebo sada protokolů standartu H.323[5], Výše zmíněné protokoly mají nejméně jeden řídící kanál využívající TCP protokol pro výměnu řídících příkazů a minimálně jeden kanál pro přenos multimediálních dat pomocí protokolů TCP nebo UDP. IP adresy a čísla portů zjistí z řídícího kanálu.

Kapitola 5

Návrh stavového firewallu

Tato kapitola popisuje návrh jednotlivých bloků statového firewallu do již existujícího bezstavového firewallu, který bude doplněn o stavovost. Návrh na implementaci stavového firewallu vychází z kapitoly 4.

Cílem implementace je upravit existující firewall tak, aby podporoval stavové filtrování paketů. Je třeba vytvořit tabulku pro uchovávání otevřených spojení a s tím související co nejjednoznačnější identifikaci spojení realizovanou pomocí hashovací funkce. V této tabulce bude docházet ke kontrole stavů jednotlivých spojení. Dále je třeba přidat kontrolu dalších položek v hlavičkách protokolů transportní vrstvy pro kontrolu stavu jednotlivých komunikací. U protokolu TCP bude docházet ke kontrole bitů SYN a ACK při navazování spojení, příznaku RST ve všech stavech komunikace a příznaky FIN a ACK během ukončování komunikace.

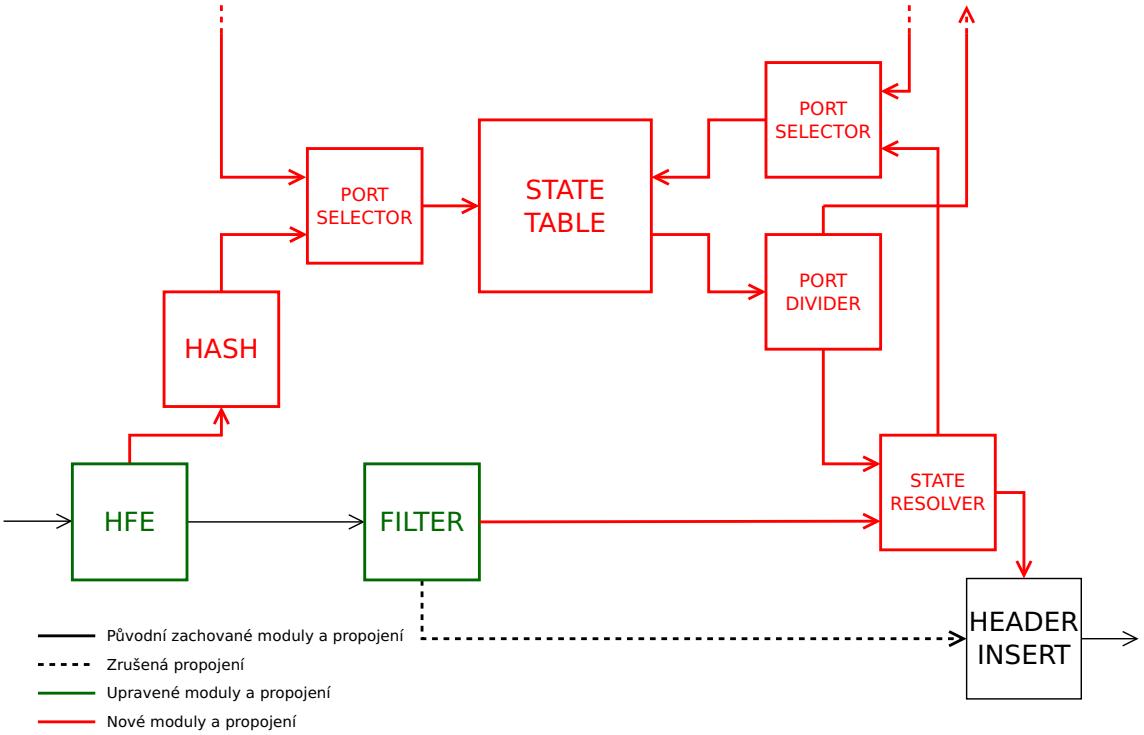
Návrh stavového firewallu bude vytvářen tak, aby bylo třeba modifikovat co nejméně již existujících modulů bezestavového firewallu. Na obrázku 5.1 lze vidět, jaké moduly bezestavového firewallu je třeba upravit a jaké moduly je třeba kompletně vytvořit.

V následujících kapitolách je popsán návrh jednotlivých komponent stavového filtrování.

5.1 Úprava jednotky *HFE-M Extractor*

Pro získávání informací z hlaviček příchozího paketu slouží moduly *HFE-M* a *HFE-M Extractor*[2]. *HFE-M* modul slouží pro detekci přítomnosti jednotlivých protokolů a offsetů hlaviček jednotlivých protokolů. Pomocí tohoto posunu je možné získat začátek hlaviček použitých přenosových protokolů. Modul *HFE-M Extractor* slouží pro extrakci jednotlivých položek z hlaviček paketu na základě výstupů z modulu *HFE-M*.

Rozhraní současného modulu *HFE-M Extractor* poskytuje z hlediska stavového filtrování užitečné informace v podobě zdrojové a cílové IP adresy verze 4 i 6 a zdrojový a cílový port. Neposkytuje však informace o příznacích TCP komunikace, které pomáhají s identifikací jejího stavu. K tomu je třeba upravit modul *HFE_M_EXTRACT_FULL* tak, aby z hlavičky TCP protokolu přečetl také příznaky dané komunikace. Přesná pozice příznaků je uvedena v [16], stejně jako posun pozice příznaků oproti prvnímu bajtu TCP hlavičky. Každý z příznaků bude reprezentován výstupním signálem s názvem odpovídajícím příznaku.



Obrázek 5.1: Přehled úprav v existujícím firewallu

5.2 Rozšíření pravidel pro stavové filtrování

Pro stavové filtrování jednotlivých komunikací je třeba modifikovat filtrovací pravidla o možnost zadání kontroly stavu pomocí nově zavedených akcí. Akce budou vypadat následovně:

CHECK Pravidlo bude sloužit pouze pro ověřování stavu komunikace. Paket vyhovující tomuto pravidlu bude moci být součástí navázané komunikace a může také komunikaci ukončit. Nemůže však provést první krok při vytváření TCP spojení. Pokud tedy paket bude mít nastaven SYN příznak a pravidlo bude obsahovat příkaz CHECK, nedojde k propuštění paketu.

UPDATE Má stejné chování jako pravidlo CHECK, ale může provést první krok při vytváření spojení. Paket s nastaveným příznakem SYN bude propuštěn a vznikne tak potřeba vytvořit také nový záznam ve stavové tabulce.

Filtrovací akce, které nebudou mít zadanou kontrolu stavu, se budou chovat stejně jako v bezestavové implementaci.

5.2.1 Paměť pro uchování pravidel

Tato rozšířená pravidla je třeba nahrát během konfigurace firewallu. Proto dojde k rozšíření paměti v rámci modulu *FILTER_RESULT_PROC*, kde jsou uloženy příslušné akce filtrovacích pravidel. Paměť akcí je realizována pomocí paměti distribuované do SLICE jednotek. Každou položku paměti je třeba rozšířit o dva bity na 25 bitů pro uchování akcí v rámci stavového filtrování. Tento formát je zvolen pro zachování existující funkčnosti, kdy 22. bit

Hodnota bitů	Význam
00	Bezestavové filtrování
01	CHECK
10	UPDATE
11	bez využití

Tabulka 5.1: Význam přidaných bitů do paměti akcí

označuje platnost vyhledané akce a ostatní bity značí samotnou akci. Přidané bity mají význam uvedený v tabulce 5.2.1.

Výstup tohoto modulu bude odpojen od modulu *HEADER_INSERT* a místo toho bude připojen k modulu *STATE_RESOLVER*, který bude popsán v kapitole 5.5. Tímto přepojením dojde k zajištění toho, aby firewall zohlednil aktuální podobu stavu komunikace.

5.2.2 Konfigurační soubory

Úpravy filtrovacích pravidel je třeba také zahrnout do konfiguračních XML[27] souborů. Konkrétně se bude jednat o úpravu výsledných akcí, kdy budou přidány volby uvedené v tabulce 5.2.1. Výjimku tvoří příkaz NO, který bude automaticky aplikován, pokud nebude zadán ani jeden z příkazů CHECK nebo UPDATE.

Akce stavového firewallu budou mít dle stavového filtrování tento formát:

- **allow** [**core**]/[**mask**] **UPDATE** značí příkaz umožňující zahájit komunikaci. Při kolizi záznamů bude používat výchozí pravidlo zadané pomocí generického typu.
- **allow** [**core**]/[**mask**] **CHECK** příkaz slouží pro kontrolu již navázané komunikace, při kolizi záznamů bude používat výchozí pravidlo zadané pomocí generického typu.
- **trim** [**length**] [**core**]/[**mask**] **UPDATE/CHECK** použije UPDATE/CHECK a oříznutí délky paketu, při chybě provede pouze oříznutí paketu.

Akce send [interface] a deny zůstanou beze změn. Změny se tedy týkají pouze obsahu tagů `<rule>`.

5.3 Identifikace spojení

Pro stavové filtrování je důležitá co nejjednoznačnější identifikace dané komunikace. Identifikátor komunikace bude sloužit jako adresa pro přístup do stavové tabulky, popsané v kapitole 5.5. Zároveň je třeba jednoduchá a rychlá implementace této identifikace, aby nedocházelo ke zdržení při výpočtu identifikátoru.

Pro tento princip přístupu ke stavové tabulce je vhodné použít datovou strukturu nazývanou tabulka s rozptýlenými položkami[7] nebo-li hashovací tabulka. Základem této techniky je princip tabulky s přímým přístupem. Každá vstupní data jsou identifikována tzv. klíčem, který popisuje charakteristiku dat. Pomocí mapovací funkce je tento klíč transformován na index položky ve stavové tabulce.

Problémem těchto tabulek je nalezení vhodné mapovací funkce tak, aby nedocházelo ke kolizi klíčů. Kolize klíčů nastává v okamžiku, kdy se dva různé klíče mapují na stejnou pozici v tabulce. V softwarové implementaci se kolize řeší pomocí index-sekvenčního přístupu

s explicitním nebo implicitním zřetězením. Tyto metody jsou však pro použití v hardwaru nevhodné, protože vedou na sekvenční, resp. index-sekvenční provádění, čímž dojde ke značnému omezení rychlosti hardwarové implementace stavové tabulky.

Další možností je použití tzv. perfektních hashovacích funkcí [4], u kterých nedochází ke kolizi klíčů. Tyto funkce se dělí podle toho, zda množina vstupních dat je proměnná nebo ne, na statické a dynamické. Z hlediska síťového provozu je třeba počítat s proměnnou množinou vstupních dat. Nalézt dynamickou perfektní hashovací funkci je velice problematické a zároveň náročné na velikost paměti.

Proto je třeba počítat s určitým počtem kolizí a případně se je snažit co nejvíce omezit nebo alespoň odhalit v rámci návrhu adresace stavové tabulky.

Při identifikaci spojení je třeba dát pozor na situaci, že zdrojová adresa v jednom směru komunikace je cílovou adresou v opačném směru komunikace, a naopak. Jelikož COMBO karta zapojená jako firewall využívá oba porty pro Ethernetovou komunikaci, je třeba vždy pro jeden port při výpočtu identifikace zaměnit zdrojovou IP adresu za cílovou a naopak. Podobně je třeba zaměnit komunikační porty. K identifikaci portu, u kterého je třeba provádět záměnu, bude sloužit již existující signál `hh_swap` použitý v modulu *HEADER_HASH*.

Z IP adres a komunikačních portů je třeba jejich kombinací vytvořit vstup modulu pro výpočet CRC-32[12]. Vstupem je 256-ti bitová posloupnost vytvořená z uvedených položek. Délka posloupnosti je zvolena s ohledem na nutnost uchovat dvě IPv6 adresy, každou o 128-mi bitech. Princip vytvoření vstupní posloupnosti bitů je naznačen na obrázku 5.2.

IPv4

source port	zeros	source IP address	source IP address	source IP address
destination port	zeros	destination IP address	destination IP address	destination IP address

IPv6

XOR source port	source IP address
XOR destination port	destination IP address

Obrázek 5.2: Vstup pro výpočet identifikace spojení a tagu

Zároveň tento obvod slouží pro přenos příznaků TCP protokolu dále do modulu stavové tabulky kvůli vyhodnocení aktuálního a identifikaci budoucího stavu komunikace. Bude také obsahovat identifikaci, zda se jedná o paket TCP komunikace, protože pro ne-TCP spojení není třeba provádět vyhledávání ve stavové tabulce.

Jelikož modul pro výpočet identifikace spojení je umístěn mezi moduly *HFE-M* a *PORT_SELECTOR*, bude třeba použít pomocný registr pro uchování informace o provedeném předání dat do modulu *PORT_SELECTOR*. Modul *HFE-M* totiž obsahuje na výstupu validní data několik taktů, ale *PORT_SELECTOR* bude provádět načítání v každém taktu hodin, kdy je nastaven validní signál. Registr bude sloužit pro detekci, že data

již byla jednou do fronty vložena.

5.4 Souběžný přístup ke stavové tabulce

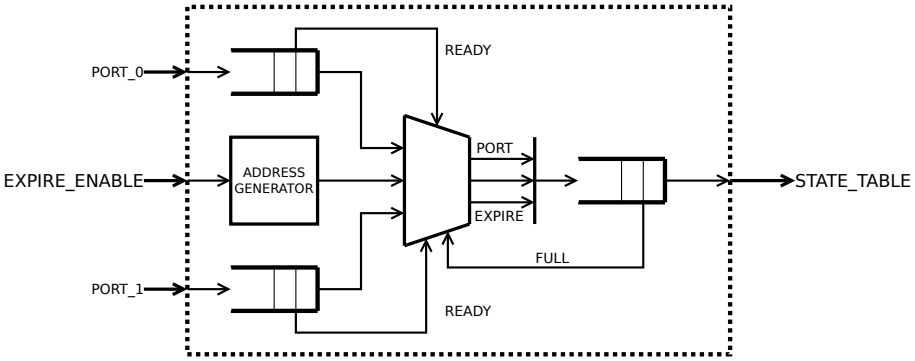
Jelikož tabulka stavů průchozích komunikací bude sdílená pro oba porty, je třeba zajistit souběžný přístup obou komunikačních kanálů ke stavové tabulce. Toho bude dosaženo pomocí fronty požadavků, kdy oba kanály budou své požadavky na čtení informací ze stavové tabulky vkládat do FIFO fronty. Požadavky pomocí fronty budou serializovány. Serializaci je třeba také implementovat při aktualizacích stavové tabulky, protože ty mohou být prováděny rovněž z obou portů.

Zároveň je třeba vytvořit modul, který informace po přečtení stavové tabulky opět rozdělí na příslušné porty. Kvůli použití front bude třeba sledovat jejich obsazenost a v případě zaplnění pozastavit průchod paketů až do doby uvolnění fronty.

5.4.1 Modul *PORT_SELECTOR*

Tento modul bude sloužit pro serializaci přístupů z různých portů při čtení a modifikaci stavové tabulky. Zároveň bude modul obsahovat generátor informací pro ověřování stáří jednotlivých záznamů. Generátor bude aktivován pouze při použití modulu v rámci čtení informací ze stavové tabulky, pro modifikaci stavové tabulky generátor není třeba použít.

Pokud bude na obou portech probíhat komunikace, kterou bude třeba kontrolovat pomocí informací ve stavové tabulce, budou se jednotlivé kanály při čtení hodnot stavové tabulky střídat. Pokud z jednoho portu nebude třeba provádět kontrolu ve stavové tabulce, budou se obsluhovat požadavky pouze toho vstupu, který kontrolu ve stavové tabulce vyžaduje. Tato situace může nastat, pokud na jednom z portů nejsou vstupní pakety. V případě prázdnosti obou vstupních front budou jednou za zvolený počet taktů generovány informace pro ověření stáří záznamů a jejich případné zneplatnění. Blokové schéma modulu lze vidět na obrázku 5.3.

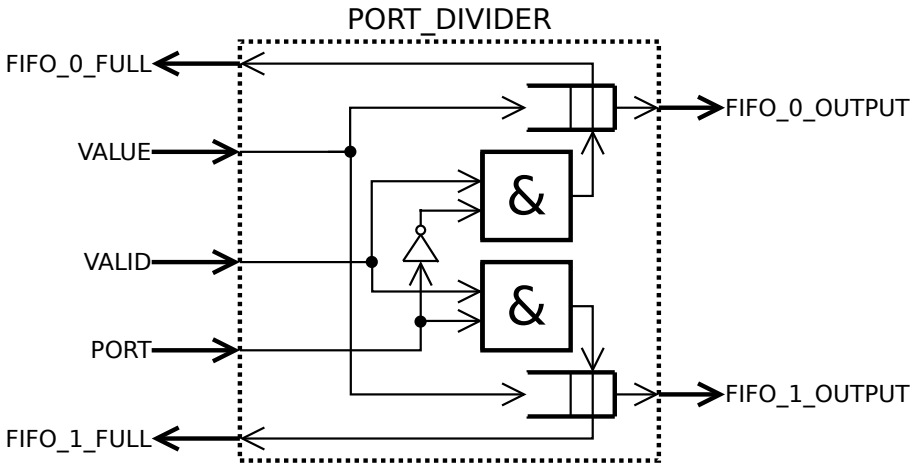


Obrázek 5.3: Modul *PORT_SELECTOR*

Jelikož stáří záznamu bude možné posuzovat v obou modulech *STATE_RESOLVER*, je při vygenerování dat pro expiraci záznamu nastaven port dle nejméně významného bitu. Stáří záznamů na sudých adresách bude tedy vyhodnocováno v rámci portu 0, liché adresy pak bude vyhodnocovat port 1.

5.4.2 Modul *PORT_DIVIDER*

Modul *PORT_DIVIDER* slouží pro rozdělení informací získaných ze stavové tabulky na jednotlivé porty. Obvod bude přijímat data ze stavové tabulky, a dle identifikace portu bude data rozdělovat do výstupních front, které budou vyrovnávat rychlost zpracování informací v modulu stavové tabulky a modulu pro vyhodnocení stavovosti *STATE_RESOLVER*. Blokové schéma modulu lze nalézt na obrázku 5.4.



Obrázek 5.4: Modul *PORT_DIVIDER*

K zápisu do příslušné výstupní fronty dojde pouze tehdy, pokud identifikace portu odpovídá dané frontě a zároveň jsou výstupní data platná. Modul bude také obsahovat signály pro signalizaci zaplnění výstupní fronty pro příslušný port. Signály budou oddělené, tudíž je možné propouštět data určená pro druhý port.

5.5 Tabulka pro uchování stavu spojení

Tabulka pro uchování stavů je důležitou součástí stavového firewallu. Obsahuje informace o jednotlivých spojeních, které procházejí skrze stavový firewall. Tabulku bude třeba realizovat pomocí víceportové paměti pro možnost současného přístupu do stavové tabulky při čtení a modifikaci záznamů.

Stavová tabulka bude obsahovat informaci o stavu komunikace, dobu poslední aktualizace reprezentovanou pomocí časového razítka a TAG pro snížení pravděpodobnosti záměny dvou různých komunikací. Adresace paměti bude realizována pomocí identifikace spojení.

5.5.1 Velikost stavové tabulky

Na velikost stavové tabulky se lze dívat z několika pohledů. Pro co nejmenší pravděpodobnosti kolizí je třeba, aby tabulka byla co největší. Čím je ale tabulka větší, tím dochází k větší spotřebě zdrojů. Proto je třeba zvolit vhodný kompromis.

S velikostí stavové tabulky souvisí velikost adresové směrnice paměti. Jelikož pro adresaci položek se používá CRC-32[12] s výstupem o velikosti 32 bitů, je maximální velikost stavové tabulky 4 GB, což je však z hlediska dostupných zdrojů nemožné splnit. Proto se pro adresaci bude používat pouze několik nejméně významných bitů[29]. Konkrétní počet bitů

bude nastavitelný pomocí generických parametrů entity a od velikosti adresové sběrnice odvozená velikost paměti.

5.5.2 Čtení a zápis informací

Během zpracování a filtrování jednoho paketu je třeba přistoupit ke stavové tabulce hned dvakrát. Poprvé při čtení informací ze stavové tabulky pro vyhodnocení výsledné akce aplikované na daný paket a podruhé při aktualizaci informace ve stavové tabulce podle nového stavu komunikace. Z tohoto důvodu bude třeba řešit situaci, kdy dojde ke čtení informace ze stavové tabulky před její aktualizací. Tím by se pro vyhodnocení využil dřívější stav komunikace, což je špatné.

Tudíž je třeba zavést mechanismus transakcí, kdy k dalšímu čtení položky nesmí dojít dříve, než je položka aktualizována předchozí transakcí. K identifikaci spuštěné transakce bude sloužit příznak spuštěné transakce umístěný v pomocné paměti o stejné velikosti, jako stavová tabulka. Pokud je tento bit nastavený, je třeba čtení ze stavové tabulky pozdržet do taktu, kdy bude příznak spuštěné transakce vynulován. Dokud bude transakce blokovat provádění, čekají všechny položky v rámci vstupní fronty.

Při čtení stavové tabulky se bude číst ta položka, na kterou ukazuje adresa získaná z údajů o spojení. Aby byla čtená položka považována za platnou, musí být nastaven bit platnosti do hodnoty logická 1 a také se musí shodovat TAG, který je tvořen jinou funkcí než identifikace komunikace, ale používá stejné vstupy. Slouží pro snížení pravděpodobnosti neodhalených kolizí.

Zápis aktualizovaných informací do stavové tabulky bude již probíhat bez kontroly bitu spuštěné transakce, protože se předpokládá, že transakce již byla spuštěna při čtení stavových informací. Rovněž nebude docházet k testování TAGu, protože ten se během spuštěné transakce nemůže měnit. Pokud při čtení informace nedošlo ke shodě TAGu, bude výsledek stavové tabulky hlášen jako kolize, takže k zápisu a tím i aktualizaci stavu nebude docházet vůbec. Aktualizaci, resp. zneplatnění záznamu bude moci způsobit také fakt, že záznamu vypršela doba platnosti.

Čtení informace

Čtení informace ze stavové tabulky bude realizováno ve dvou krocích. V prvním kroku se pomocí identifikace spojení přečte indexovaná položka z paměti stavové tabulky i z paměti transakcí. Pomocí informace z paměti transakcí dojde k určení, zda je položka stavové tabulky používána, nebo ne. Čtení ze stavové tabulky bude prováděno pouze při ověřování stáří záznamu nebo vyhledávání stavu TCP komunikace. V opačném případě se čtení provádět nebude, a do výstupní fronty se chybějící položky vynulují.

Pokud není položka používána, dojde ve druhém kroku k zápisu informace o spuštěné transakci do paměti transakcí. Informace získané ze stavové tabulky doplní hodnoty extrahované z hlavičky paketu a společně budou odeslány do obvodu *PORT_DIVIDER* pro rozdělení na příslušné porty.

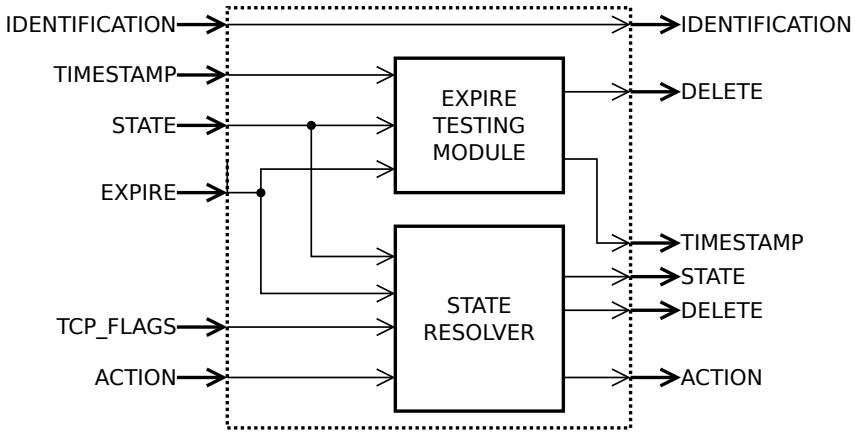
Zápis informace

Obvod pro zápis již nemusí číst položky ze stavové tabulky ani z tabulky spuštěných transakcí, proto v jednom kroku provede současně zápis aktualizované informace do stavové tabulky a také zapíše ukončení transakce do tabulky spuštěných transakcí.

5.6 Obvod pro vyhodnocení stavu komunikace

Tento obvod zajišťuje spojení existující části bezestavového firewallu s moduly zajišťujícími stavové filtrování. Jak je vidět na obrázku 5.1, modul *STATE_RESOLVER* je zapojen mezi existující moduly *FILTER* a *HEADER_INSERT*. Vyhodnocení probíhá na základě akce při splnění podmínky filtrovacího pravidla a stavu komunikace uvedeného ve stavové tabulce. Výstupem obvodu je poté akce v původním formátu bezestavového filtrování, tedy s odstraněnými bity, které budou přidány pro definování akce stavového filtrování.

Vstupy a výstupy modulu ukazuje obrázek 5.5.



Obrázek 5.5: Modul *STATE_RESOLVER*

Činnost obvodu se liší podle toho, pokud informace ze stavové tabulky slouží pro ověření stáří záznamu, nebo pro vyhodnocení současné komunikace a případné modifikaci stavové tabulky. Hlavním rozdílem je čtení informací ze vstupních front. Fronta obsahující stavové informace bude čtena v každém taktu, ve kterém budou k dispozici data ve frontě. Tato data budou obsahovat jednobitovou informaci o tom, zda se týkají ověření stáří záznamu, nebo informací potřebných pro stavové filtrování komunikace. Ve druhém případě je třeba mít k dispozici také informaci o příslušném filtrovacím pravidlu a akci.

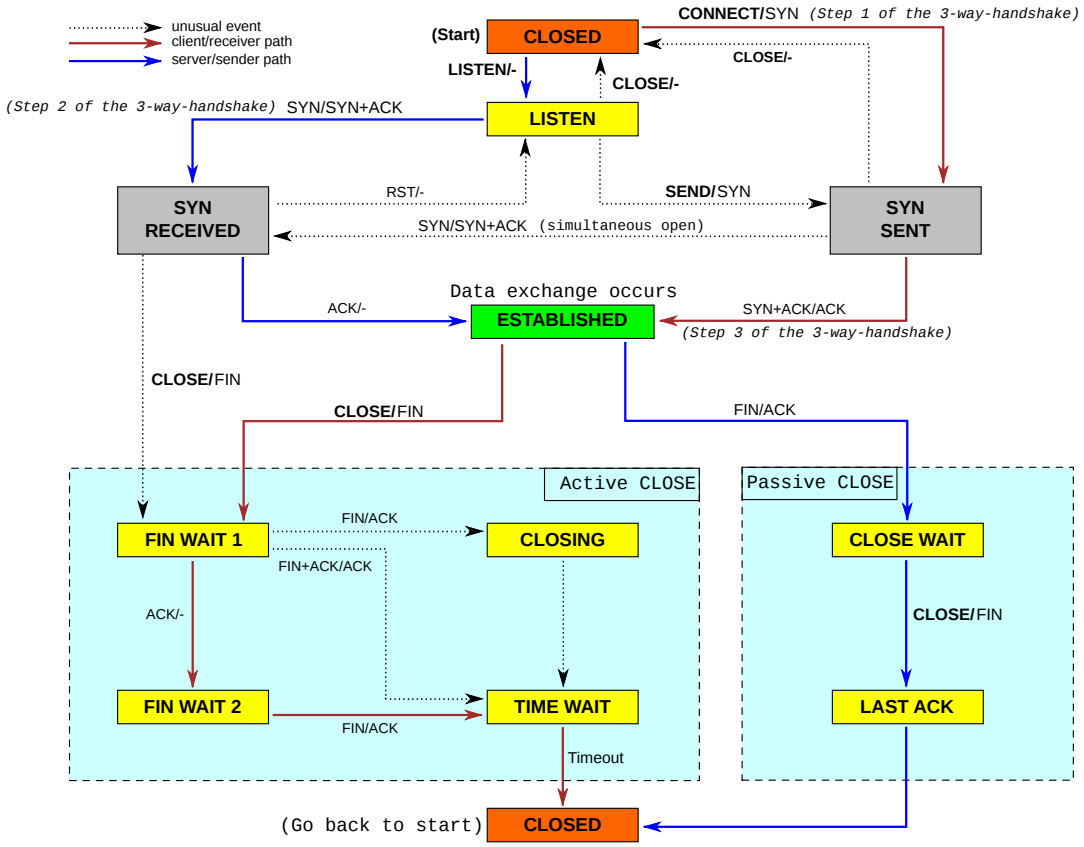
Obvod zároveň obsahuje vstup pro identifikaci spojení a příznaky TCP protokolu kvůli uchování těchto informací. Užití těchto informací je vysvětleno v podkapitole 5.6.1. Jelikož je třeba řešit odhalené kolize ve stavové tabulce, bude modul obsahovat vstup, který říká, jakou výslednou akci při kolizi použít. Tento vstup bude třeba nastavit při syntéze firewallu.

5.6.1 Modul *STATERESOLVER_LOGIC*

Tento modul bude sloužit pro vyhodnocení výsledné akce, která bude provedena se zkoumaným paketem na základě akce uvedené ve filtrovacím pravidlu a stavu komunikace ve stavové tabulce.

Vstupem modulu jsou informace ze stavové tabulky, příznaky zkoumané TCP hlavičky a akce získaná z filtrovacích pravidel pro daný paket. Obvod pro vyhodnocení bude realizován pomocí kombinační logiky, která podle zadaných podmínek provede vyhodnocení výsledné akce a zároveň vytvoří informace, kterými je třeba aktualizovat stavovou tabulku.

Na obrázku 5.6 lze vidět diagram reprezentující průběh TCP spojení.



Obrázek 5.6: Diagram TCP komnunikace, převzato z [28]

Navázání nového spojení

Navazovat spojení bude možné pouze tehdy, pokud se bude jednat o komunikaci využívající TCP protokol, v TCP hlavičce paketu bude nastaven pouze příznak SYN a výsledná akce pro filtrování bude obsahovat pokyn pro vytvoření komunikace. Zároveň je třeba ze stavové tabulky získat informaci, že odpovídající položka ve stavové tabulce je volná, jinak je třeba nahlásit kolizi. Při kolizi ve stavové tabulce je postupováno podle nastavené výchozí akce a ve stavové tabulce k vytvoření nového záznamu nedojde.

Poté dochází ke sledování dalších paketů navazujících TCP spojení. Stav komunikace obsahuje informaci, že posledně přijatý paket obsahoval nastavený SYN příznak. Pokud dorazí další paket se stejnou identifikací, musí obsahovat nastavené příznaky SYN a ACK, na výsledné akci (kontroluj nebo aktualizuj) v tomto okamžiku již nezáleží. Jiné pakety v tomto případě propuštěny nejsou. Pokud třetí obdržený paket obsahuje nastavený příznak ACK, dojde k ustanovení komunikace.

Kontrola spojení

Ke kontrole komunikace dochází v obou případech stavové akce. V případě nalezení záznamu ve stavové tabulce je výsledná akce vyhodnocena pomocí aktuálního stavu komunikace. Při navázané komunikaci dochází k provozu paketů obsahujících data, které nemají nastavený

žádný příznak, a paketů sloužících pro potvrzení přijetí (nastaven příznak ACK). TCP spojení umožňuje, že pomocí datového paketu dojde k potvrzení předchozího paketu v opačném směru.

Uzavření spojení

K ukončování TCP spojení je použit čtyřcestný handshake. TCP spojení je plně duplexní, takže i pro ukončení spojení je třeba uzavřít oba toky dat. Strana komunikace, která chce spojení ukončit, zašle paket s nastaveným příznakem FIN. Spojení však v rámci stavového firewallu musí zůstat navázáno, protože druhá strana může ještě odesílat data. Odpovědí na příznak FIN je paket s příznakem ACK. Spojení zůstane navázané do té doby, než druhá strana také odešle paket s příznakem FIN a obdrží jako odpověď paket s nastaveným příznakem ACK. Proto je třeba s tímto způsobem ukončování komunikace počítat i ve stavové tabulce.

Restart spojení

Pokud během několika pokusů o znovuzaslání nedojde k přijmu potvrzujícího paketu, může strana odesílatele zaslat paket s nastaveným příznakem RST. Pokud protistrana přijme paket s nastaveným příznakem RST, dojde ke zrušení spojení. Proto je třeba informaci o navázaném spojení smazat i ve stavové tabulce.

5.6.2 Modul *STATERESOLVER_EXPIRE*

Tento modul bude mít dvě využití dle toho, zda se bude kontrolovat stáří záznamu nebo bude prováděno filtrování paketu. Při kontrole stáří záznamu provede vygenerování aktuálního časového razítka a provede porovnání s kontrolovaným časovým razítkem. Zároveň bude vstupem obvodu i aktuální stav dané komunikace kvůli různému omezení na stáří záznamů. Výstupem bude příznak, zda je možné záznam v tabulce ponechat, nebo provést jeho zneplatnění.

Simulaci přesného času bude zajišťovat čítač hodinových taktů. Horní hranice čítání bude nastavena podle hodinové frekvence celého obvodu. Po překročení horní hranice bude čítač taktů vynulován a dojde k inkrementaci signálu sloužícího jako časové razítko.

Doba platnosti časového razítka se bude lišit podle toho, zda se jedná o navazované nebo ukončované spojení, nebo spojení navázané.

5.6.3 Výstupy modulu

Tento modul bude obsahovat dvě množiny výstupních signálů. První množina výstupů bude tvořena rozhraním, které je shodné s výstupem modulu *FILTER* v původní implementaci. Modul *HEADER_INSERT* tedy zůstane beze změn.

Druhá množina výstupů bude tvořit rozhraní pro aktualizaci informací ve stavové tabulce. Rozhraní bude obsahovat sběrnici pro přenos časového razítka, příznaků, nového stavu komunikace a její identifikaci. Toto rozhraní bude přivedeno na vstupy modulu *PORT_SELECTOR*, který bude sloužit pro serializaci zápisů do stavové tabulky. Data budou zasílána pouze v případě, že zkoumaný paket byl součástí TCP komunikace, nebo záznam obsahuje výsledek vyhodnocení stáří záznamu. V opačných případech není třeba aktualizovat tabulku, protože nebude docházet ani ke spuštění transakce.

Kapitola 6

Implementace

Tato část popisuje implementaci rozšíření jednotlivých modulů a také implementaci nových modulů. Kapitola obsahuje implementaci vybraných modulů a detailně popisuje komunikaci jednotlivých modulů, včetně komunikačních protokolů mezi jednotlivými dvojicemi modulů. Věnuje se také syntéze navrženého stavového firewallu do FPGA obvodu a počet použitých zdrojů.

6.1 Stavová tabulka *STATE_TABLE*

Stavová tabulka a paměť transakcí využívá paměť BlockRAM uvnitř FPGA čipu pomocí modulu *DP_BMEM*, který je součástí platformy NetCOPE. Jedná se o dvouportovou paměť, takže bude moci současně probíhat zápis informace a čtení. Kvůli urychlení systému transakcí jsou obě paměti nastaveny tak, aby nejprve došlo k zápisu a až poté ke čtení informace. Pokud bude iniciováno čtení i zápis stejné položky ve shodném taktu, dojde nejprve k zápisu a tím i k ukončení transakce a až poté ke čtení stejné položky včetně informace, že je transakce ukončena.

Položka stavové tabulky obsahuje informace o stavu komunikace, jejíž identifikace odpovídá adrese dané položky. Struktura položky je uvedena v tabulce 6.1. Položky, u kterých je bitová šířka uvedena jako generická, mají v závorce uvedeny bitové šířky použité pro současnou implementaci. První položka shora tabulky se nachází na nejvíce významných bitech[30] jednoho záznamu v paměti.

Položka reprezentující poslední známý stav komunikace v sobě skrývá nastavené příznaky při nově vytvářeném nebo ukončovaném spojení či informaci o navázaném spojení. Reprezentace jednotlivých stavů je uvedena v tabulce 6.2. Hodnota příslušných bitů má tento význam pouze v případě, že je záznam platný (nastavený bit platnosti).

Stavová tabulka v současné implementaci obsahuje celkem 1024 položek, je tedy ad-

Položka	**Šířka [b]**	**Význam**
VALID	1	Platnost záznamu
TAG	generická (8)	Pomocná položka pro odhalení kolize záznamů
STATE	3	Poslední známý stav komunikace
TIMESTAMP	generická (10)	Časové razítko posledního použití záznamu

Tabulka 6.1: Obsah položek stavové tabulky

Hodnota	Význam položky, nastavené příznaky
000	Neznámý stav komunikace, reprezentuje nenalezený záznam
001	Nastavený příznak SYN
010	Nastavené příznaky SYN a ACK
011	Kolize TAGů, využito ve výstupní informaci o stavu
100	Navázané spojení
101	Nastavený příznak FIN z portu 0
110	Nastavené příznaky FIN z portu 1
111	Nastavené příznaky FIN z obou portů

Tabulka 6.2: Reprezentace stavu spojení ve stavové tabulce

resována 10 bity. Je to maximální použitelná hodnota vyhovující omezením na časování obvodu.

6.1.1 Čtení hodnoty a spuštění transakce

Čtení ze stavové tabulky a následné spouštění transakcí se liší podle toho, jaká data jsou právě na začátku vstupní fronty obsahující identifikaci komunikace. Detailně je chování pro jednotlivé situace popsáno v následujících odstavcích.

Kontrola stáří záznamů

Pokud data obsahují příkaz ke kontrole stáří záznamu, dojde ke čtení jak stavové tabulky, tak tabulky spuštěných transakcí. Na výstup jsou však umístěna data ze stavové tabulky pouze tehdy, pokud jsou platná a zároveň pokud není nad daty spuštěna transakce. Výstupní informace obsahují adresu odpovídající identifikaci spojení, poslední známý stav komunikace a časové razítko. Zároveň je třeba provést spuštění transakce, protože později mohou být data ve stavové tabulce zneplatněna.

Jinak nemá smysl provádět kontrolu stáří záznamu, protože pro neplatný záznam není třeba ověřovat jeho stáří a pro záznam se spuštěnou transakcí bude během několika taktů hodin aktualizován. Proto v těchto dvou případech nejsou na výstup umístěny žádné informace.

TCP spojení

TCP spojení je identifikováno nastaveným příznakem TCP. V tomto případě je třeba číst informace z obou tabulek. Poté je třeba podle hodnoty z tabulky transakcí rozlišit další chování.

Pokud není nad požadovanými daty spuštěna transakce, dojde k zápisu dat na výstup stavové tabulky a ke spuštění transakce nastavením hodnoty v paměti transakcí. Výstupní data obsahují informace jednak ze vstupních dat (identifikace spojení, tag, příznaky a výběr portu), tak informace přečtené ze stavové tabulky (poslední známý stav komunikace). Stav komunikace může být upraven, pokud nedojde ke shodě TAGů. V tomto případě je stav komunikace nastaven na hodnotu značící kolizi ve stavové tabulce. Další možnou úpravou stavu je nalezení neplatného záznamu, u kterého je stav komunikace nastaven na hodnotu záznam nenalezen. Uvedené hodnoty vycházejí z tabulky 6.2.

Problémem je situace, kdy bude v jednom taktu čten záznam ze stavové tabulky a zároveň ve stejném taktu transakce ukončena. Toto však umožňuje řešit modul *DP_BMEM*, u kterého lze nastavit upřednostňovanou operaci. Pro tento případ je vhodné upřednostnit operaci zápisu, protože při čtení bude vidět ukončená transakce i ve stejném taktu, jako vznikl požadavek na čtení.

Ostatní spojení

Pro ostatní spojení (příznak TCP je nulován) dojde ihned při obdržení odeslání vstupních dat na výstup. Jelikož se z hlediska implementovaného stavového firewallu nejedná o spojení se stavem, není třeba číst informace ze stavové tabulky ani spouštět transakci, čímž zároveň dojde k urychlení filtrování nestavových spojení. Informace musí být na výstup umístěna, protože je třeba doručit vstupní informace dále do modulu *STATE_RESOLVER*.

6.2 Logika modulu *STATERESOLVER_LOGIC*

Tvoří součást modulu pro vyhodnocení komunikace. Na základě posledního známého stavu TCP komunikace, nastavených příznaků v TCP hlavičce paketu a akce vyplývající z filtrovacích pravidel. Protože vyhodnocení komunikace probíhá vždy pouze pro aktuálně zpracovávaný paket, je obvod realizován kombinační logikou, která zajišťuje jak vytvoření nového stavu, tak vygenerování původní akce pro další část obvodu.

Obvod provádí, dle posledně známého stavu, kontrolu nastavených příznaků. Pokud je nastavena nepovolená kombinace příznaků vzhledem ke stavu komunikace, je paket ihned zahozen. Ke kontrole dochází při každém stavu TCP komunikace, tedy při navazování nového spojení, existenci ustáleného spojení i při ukončování spojení.

Vyjímku tvoří příchod paketu s nastaveným příznakem RST. V tomto případě je předpokládáno restartování komunikace, a proto je provedeno zneplatnění záznamu ve stavové tabulce.

Stav komunikace je sledován pouze při akcích allow a trim, akce send a deny vytvoření záznamu o stavu komunikace nezpůsobí. Obvod předpokládá, že je vždy nějaké pravidlo nalezeno, tedy při nenakonfigurované paměti pravidel je pro všechny komunikace nalezeno pravidlo allow bez stavového filtrování. Při nahrané konfiguraci je třeba pro každé rozhraní uvést pravidlo s nejmenší prioritou, které bude použito vždy, pokud komunikace nevyhovuje ostatním pravidlům s vyšší prioritou.

6.2.1 Navazování nového TCP spojení

Při navazování spojení dochází ke kontrole příznaků jednotlivých paketů. Komunikaci může zahájit pouze ten paket, pro který filtrovací akce obsahuje nastavený příznak pro vytvoření nové komunikace. Zároveň tento paket musí obsahovat nastavený pouze příznak SYN. Druhý paket v pořadí může buď obsahovat nastavené příznaky SYN a ACK nebo pouze příznak SYN v případě, že na první paket zahajující komunikaci protistrana nereagovala.

Stejná situace může nastat i při přijmu třetího paketu, který může buď obsahovat příznak ACK nebo oba příznaky SYN a ACK. Proto je třeba při navazování spojení počítat s variantami, že může firewallem projít několik paketů se stejně nastavenými příznaky. Pokud třetí paket obsahuje pouze nastavený příznak SYN, předpokládá se neúspěšné navázání vlivem chybějícího třetího paketu s příznakem ACK. Proto v tomto případě dojde ke zrušení předchozího stavu komunikace a novým stavem bude pouze stav reprezentující průchod

paketu se SYN příznakem.

6.2.2 Ustálené spojení

Během ustáleného spojení jsou povoleny pakety bez nastavených příkazů obsahující aplikační data nebo pakety s nastaveným příznakem ACK. Nejsou tedy přípustné pakety s nastaveným příznakem SYN. Takové jsou ihned zahazovány jako nevyhovující. Při příchodu prvního paketu s nastaveným příznakem FIN přechází komunikace do fáze ukončování, která je popsána dále.

6.2.3 Ukončování spojení

Stavový firewall považuje komunikaci ve fázi ukončování spojení od doby průchodu prvního paketu s nastaveným příznakem FIN. Jak je uvedeno v kapitole 5.6.1, je TCP komunikace považována za dvě jednosměrné komunikace. Z tohoto důvodu je třeba sledovat, zda došlo k průchodu paketu s nastaveným příznakem FIN v obou směrech komunikace. Proto je třeba při vyhodnocování stavu odlišit, jakým portem paket s nastaveným FIN příznakem vstoupil do stavového firewallu.

Toto umožňuje komunikovat v ještě neukončeném směru a také řešit situaci, kdy se paket s nastaveným FIN příznakem objeví vícekrát (např. pokud na některé pakety FIN nedorazila odpověď) na stejném portu. Pokud již přišel příznak FIN z obou směrů komunikace, je povolen pouze průchod posledního paketu s nastaveným příznakem ACK. Po jeho průchodu je komunikace považována za uzavřenou a příslušný záznam ve stavové tabulce je zneplatněn.

Tento systém sledování stavu komunikace při jejím ukončování řeší i další možné ukončení TCP komunikace, čili poradí si i s jiným ukončováním, než je běžný čtyřcestný handshake. Další možnosti ukončení komunikace lze vidět na obrázku 5.6.

6.3 Komunikace jednotlivých modulů

Kvůli vyrovnání zpoždění jednotlivých modulů je komunikace realizována pomocí front FIFO. Pro implementaci stavového firewallu je dostatečně vyhovující jednoduchá fronta *FIFO_STATUS*[23], která obsahuje povolovací vstupy pro čtení ze začátku fronty a zápis na konec fronty. K zápisu položky na konec fronty dojde, pokud je nastaven signál pro zápis a zároveň se objeví nástupná hrana hodinového signálu. Čtení je pak iniciováno pomocí nastavení příslušného signálu, hodinový signál na čtení nemá vliv.

6.3.1 Komunikace mezi *STATE_TABLE* a *STATE_RESOLVER*

Komunikační fronta mezi moduly *STATE_TABLE* a *STATE_RESOLVER* obsahuje informace potřebné pro vyhodnocení stavu komunikace nebo ověření stáří záznamu. Položka fronty vždy obsahuje adresu do stavové tabulky, stav komunikace a příznak signalizující význam položky (expirace nebo stav komunikace). Ostatní bity položky budou sdílené pro časové razítko v případě expirace a pro stav komunikace a příznaky vyčtené z hlavičky zkoumaného paketu. Proto je třeba kontrolovat minimální velikost položky fronty, aby fronta mohla obsahovat veškeré informace pro vyhodnocení stavu komunikace. Protokol komunikační fronty je uveden v tabulkách 6.3 a 6.4. Bitová šířka položek fronty je dána maximem z bitové šířky časového razítka a součtu počtu příznaků a bitové šířky TAGu.

Název položky	Počet bitů
INDEX	generický(10)
STATE	3
TAG	generický (8)
TCP	1
SYN	1
ACK	1
RST	1
FIN	1
PORT	1
EXPIRE = 0	1

Tabulka 6.3: Význam bitů při přenosu stavu komunikace

Název položky	Počet bitů
INDEX	generický(10)
STATE	3
TIMESTAMP	generický (10)
PORT	1
EXPIRE = 1	1

Tabulka 6.4: Význam bitů při přenosu časového razítka

Pro zpětnou modifikaci stavové tabulky dle změny stavu komunikace je třeba vytvořit další komunikační kanál, který bude tvořen sběrnicí. Sběrnice je přímo připojena ke stavové tabulce a obsahuje jednobitový signál, který řídí zápis aktualizace stavu komunikace do stavové tabulky. Identifikátor portu je po rozdělení dat na příslušné porty vyjmut.

6.3.2 Fronty pro modul *PORT_SELECTOR*

Modul obsahuje duplikovanou množinu vstupů, přičemž každá množina slouží pro příslušný vstupní port. Jelikož při serializaci může dojít k tomu, že v jednom taktu budou na obou množinách užitečné informace, je třeba tyto informace umístit do vyrovnávacích front. Každá množina vstupů má vlastní vyrovnávací frontu. Jelikož stejná situace může nastat na výstupu, je třeba umístit frontu i na výstup obvodu. Význam položek jednotlivých front se liší podle toho, zda modul slouží pro serializaci čtení ze stavové tabulky nebo zápis do stavové tabulky. Jelikož jsou různá i rozhraní dle způsobu použití, vznikly dva moduly obsahující modul *PORT_SELECTOR*. Tyto moduly tvoří obálku a poskytují rozhraní dle způsobu využití. Zároveň provádějí transformaci vstupů na položky vstupní fronty.

Informace pro čtení stavu

Vstupní i výstupní fronty mají téměř stejný obsah položek. Výstupní fronta je pouze doplněna o signál identifikující vstupní port pro pozdější rozdělení informací. Tabulka 6.5 popisuje význam položky ve vstupní i výstupní frontě. Pro konverzi jednotlivých vstupních informací z hlaviček paketů slouží modul *PORT_SELECTOR_READ*, který uvnitř obsahuje modul *PORT_SELECTOR*.

Informace pro aktualizaci stavu

Obvod pro serializaci zápisů aktualizací síťových komunikací musí obsahovat informace potřebné pro modifikaci stavu. Význam obsahu položky fronty při použití modulu pro aktualizaci stavové tabulky je uveden v tabulce 6.6, přičemž konverzi vstupních signálů na položku fronty zajišťuje modul *PORT_SELECTOR_WRITE*. Popis možných hodnot položky `NEWSTATE` vychází z tabulky 6.2.

V každém taktu, kdy jsou k dispozici data pro aktualizaci stavové tabulky ve výstupní frontě modulu *PORT_SELECTOR_WRITE*, dochází k ukončení spuštěné transakce v ta-

Název položky	Bity vstupní	Bity výstupní	Význam
INDEX	generický (10)	generický (10)	Adresa položky ve stavové tabulce
TAG	generický (8)	generický (8)	TAG pro uložení do stavové tabulky
TCP	1	1	Informace o tom, jestli se data týkají TCP paketu
SYN	1	1	Příznak SYN protokolu TCP
ACK	1	1	Příznak ACK protokolu TCP
FIN	1	1	Příznak FIN protokolu TCP
RST	1	1	Příznak RST protokolu TCP
PORT	-	1	Výběr jednoho ze dvou vstupních portů
EXPIRE	-	1	Pokud je příznak nastaven, položka obsahuje pouze data pro expiraci

Tabulka 6.5: Formát položky pro čtení ze stavové tabulky

Název signálu	Bity vstupní i výstupní	Význam
`MODIFY`	1	Nastaven, pokud má dojít k přepsání záznamu, jinak se pouze ukončí transakce
`INDEX`	generický (10)	Adresa položky ve stavové tabulce
`TAG`	generický (8)	Adresa položky ve stavové tabulce
`TIMESTAMP`	generický (10)	Časové razítko záznamu v položce
`NEWSTATE`	3	Nový stav komunikace
`DELETE`	1	Nastaven, pokud má dojít ke zneplatnění záznamu

Tabulka 6.6: Rozhraní pro aktualizaci stavové tabulky

bulce transakcí. Zápis do stavové tabulky je proveden pouze v případě, že aktuální položka na začátku fronty obsahuje informace o aktualizaci záznamu. V tomto případě dojde z přenesených informací k rekonstrukci celé položky stavové tabulky s následnou aktualizací. Aktualizace se provádí i v případě mazání položky ze stavové tabulky, kdy je využita pouze hodnota signálu `DELETE`. Jeho negací získáme bit označující platnost záznamu. Při zneplatnění položky tabulky jsou ostatní bity nevýznamné.

6.4 Software pro nahrávání pravidel

Součástí existující implementace bezestavového firewallu je software `hanicctl`[25], který slouží pro ovládání bezestavového firewallu. Pro účely této práce je důležité, že software provádí načtení filtrovacích pravidel zadaných ve vstupním XML[27] souboru, syntaktickou a sémantickou kontrolu těchto pravidel a nahrání pravidel do paměťových struktur použitých pro vyhledávání filtrovacích pravidel a tím i příslušných akcí. Úprava na stavové filtrování se týká paměti pro uchování akcí, proto je třeba provést také úpravu pro funkce zajišťující nahrávání výsledných akcí.

Úprava programu spočívala pouze v doplnění analýzy XML souboru obsahujícího konfiguraci firewallu. Změna se týká doplnění funkce `parse_action`, ve které jsou zpracovávány akce v závislosti na filtrovacích pravidlech. Dle návrhu 5.2.2 úpravy syntaxe konfiguračního souboru pro filtr je doplněna kontrola zadání akcí CHECK a UPDATE pro stavové filtrování. Tyto příkazy způsobí nastavení bitů 23 a 24 dle tabulky 5.2.1. Jelikož filtr umožňuje pro každý port definovat vlastní filtrovací pravidla, je třeba, aby filtrovací pravidlo obsahující podmínku pro stav komunikace bylo zadáno v konfiguraci obou portů.

Software pro konfiguraci umožňuje kromě adres konkrétního počítače zadávat také rozsah adres pomocí IP adresy a masky sítě. Pokud bude nějaké filtrovací pravidlo takto zadáno, je třeba pro každou komunikaci, která tomuto pravidlu vyhovuje a zároveň pravidlo obsahuje příkaz ke stavovému filtrování, je třeba v této množině IP adres sledovat každou komunikaci zvlášť. Jelikož ale identifikace spojení v hardwaru používá vždy konkrétní IP adresy, je tato situace vyřešena.

6.5 Syntéza do FPGA

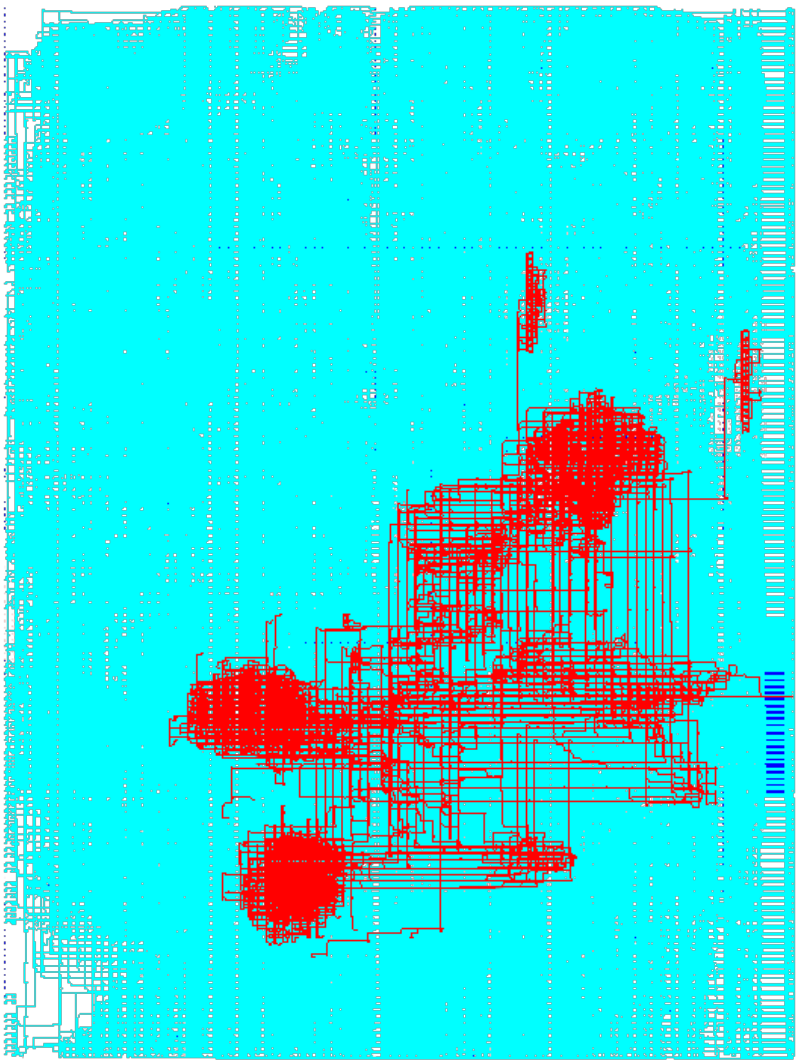
Výsledná implementace stavového firewallu musí být syntetizovatelná do FPGA obvodu, který je na kartě k dispozici, proto je třeba při vytváření dbát na množství dostupných zdrojů a také splnit požadovaná omezení na počet zdrojů a rychlost stavového filtrování. Jelikož úpravy stavového firewallu nezasahovaly do rozhraní celého firewallu, není třeba upravovat mapování vstupů nebo výstupů. Obrázky 6.1 a 6.2 znázorňuje mapování komponent a signálů stavového firewallu do cílové FPGA technologie. Nově vytvořené komponenty jsou barveny červeně.

V tabulce 6.7 je uveden počet použitých zdrojů s porovnáním vůči původní implementaci bezestavového firewallu. Zároveň dochází k ověření, zda implementovaný stavový firewall vyhovuje všem omezujícím podmínkám[33]. Výsledný obvod pracuje na frekvenci 125 MHz, což je shodná frekvence s původní implementací.

Na použitém počtu SLICEs a BlockRAMs je vidět nárůst způsobený doimplementováním stavového filtrování. Šířka identifikace je 10 b, časového razítka 10 b a TAGu 8 b. Položka stavové tabulky má tedy celkem 22 bitů včetně popisu stavu komunikace. Pro 1024 položek stavové tabulky a tabulky transakcí je jejich velikost 23 Kib, což odpovídá nárůstu

Obrázek 6.1: Mapování komponent stavového firewallu do cílové technologie

dvou alokovaných BlockRAM pamětí. Nárůst alokované paměti je však pouze 18 Kib. Je to způsobeno optimalizacemi při syntéze stavového firewallu. Přesný rozpis alokovaných zdrojů je uveden v příloze B.



Obrázek 6.2: Mapování signálů stavového firewallu do cílové technologie

Položka	Dostupné	Bezestavový	Stavový	Nárůst
Number of Slices	24320	17176 (70%)	17827 (73%)	651 (+3%)
Number of Slice Registers	97280	33906 (34%)	34335 (35%)	429 (+1%)
Number of Slice LUTS	97280	40187 (41%)	42713 (43%)	2526 (+2%)
Sum of BlockRAMs	424	227 (54%)	228(54%)	1 (0%)

Tabulka 6.7: Počet alokovaných zdrojů a porovnání s původní implementací

Kapitola 7

Testování

Tato kapitola popisuje metody ověření správné funkčnosti stavového filtrování. Zároveň také obsahuje popis testování již implementované bezestavové části pro ověření, zda-li úpravami existujících modulů nedošlo k jejich nesprávné funkci. Testování stavového firewallu probíhalo simulací pomocí programu ModelSim[1], a také testováním v reálném hardwaru. Jelikož se jedná o úpravu již existujících částí, je třeba provést úpravu také v pomocných simulačních obvodech (tzv. testbench).

7.1 Simulace upravených modulů

Při testování upravovaných modulů je třeba dbát jak na správnou funkčnost nových částí, tak také na neporušení správně fungujících původních částí modulu.

7.1.1 Modul *HFE-M*

Testování modulu HFE-M se odehrávalo v rámci simulace, kdy na vstup modulu je odesláno několik TCP paketů s cílem otestovat extrakci jednotlivých příznaků z hlavičky analyzovaného TCP paketu. Pro tuto příležitost byly vytvořeny pakety s vhodně nastavenými příznaky. Pro příznaky SYN, ACK a FIN jsou testovány všechny jejich možné kombinace, příznaky PSH, RST a URG jsou otestovány pouze samostatně. Při testování je také třeba sledovat hodnotu signálu udávajícího platnost extrakce příznaků. Pro tento případ byl vytvořen další paket, který sice má nastavené příznaky jako v protokolu TCP, ale v položce `protocol` je uvedena hodnota pro UDP protokol.

7.1.2 Modul *FILTER*

U modulu pro vyhledávání filtrovacích pravidel došlo k rozšíření informace o prováděné akci na základě vyhovujícího filtrovacího pravidla. Proto je nutné otestovat správné nahrání pravidel rozšířených o příkazy pro stavové filtrování a následou reakci při vyhodnocování výsledných akcí.

7.2 Simulace nových modulů

K testování nově vytvořených modulů posloužil opět program ModelSim, takže simulace probíhala obdobně jako v kapitole popisující testování upravovaných modulů. Zároveň je

[1] http://www.model.com/

žádoucí vytvořit testbench pro každý nově vytvořený modul. Pokud je modul složen z více jednodušších modulů, je testbench vytvořen z těchto dílčích modulů pro lepší možnost sledování hodnot jednotlivých signálů.

7.2.1 Stavová tabulka

Při testování stavové tabulky je třeba ověřit správnost implementace při práci s oběma porty paměti. Jeden port paměti slouží pro čtení informací ze stavové tabulky, druhý pro aktualizaci informací. Jelikož stavová tabulka je při průchodu paketů čtena i modifikována současně, je nutné provést současně také testování přístupu ke stavové tabulce. Součástí testování je také sledování spouštění a ukončování transakcí v paměti transakcí.

První část testování spočívala v použití různých druhů informací obsahujících identifikaci spojení, tedy informací pro čtení stavu komunikace ze stavové tabulky. Testovací vstupy obsahovaly identifikaci TCP a ne-TCP komunikace, příkaz ke kontrole stáří validního i nevalidního záznamu a identifikace, ve které nedojde ke shodě TAGu s TAGem uloženým ve stavové tabulce.

Druhá část spočívala v ověření správného zápisu aktualizovaných informací do stavové tabulky. Jednalo se o změnu stavu komunikace, vytvoření nového záznamu ve stavové tabulce a zneplatnění existujícího záznamu.

7.2.2 Vyhodnocení komunikace

Obvod pro vyhodnocení má různé chování podle toho, zda vstupní data získaná ze stavové tabulky obsahují data potřebná k ověření stáří záznamu nebo data pro filtrování komunikace. Data pro filtrování komunikace mohou obsahovat jak informace ze stavové tabulky v případě paketů TCP komunikace, nebo pouze informaci o tom, že paket není součástí žádné TCP komunikace.

Důležité je provést testování čtení ze vstupních front. Z fronty přenášející informace ze stavové tabulky je třeba číst v každém taktu, ve kterém jsou k dispozici data. Z fronty akcí je pak třeba číst pouze pokud se informace ze stavové tabulky týkají filtrování paketů. Dle typu vstupních dat přijatých ze stavové tabulky pak jsou aktivovány příslušné obvody pro vyhodnocení výsledné akce a výpočet nového stavu komunikace a obvod pro kontrolu stáří záznamu. Při testech je třeba vytvořit vstupní data tak, aby došlo k vyzkoušení všech možných stavů tohoto obvodu.

Také je třeba otestovat, že vyhodnocování podle stavu komunikace je možné pouze pokud to povolují filtrovací pravidla, jinak bude chování firewallu bezestavové.

Jelikož tento obvod uvnitř obsahuje dva další moduly, je i pro ně vytvořen testbench, který ověřuje správnou činnost modulu pro výpočet nového stavu a také modulu pro detekci příliš starého záznamu.

7.2.3 Serializace dotazů a aktualizací

Jelikož moduly *PORT_SELECTOR_READ* a *PORT_SELECTOR_WRITE* tvoří pouze obálku zajišťující konverzi vstupních signálů na formát položky front, stačí provést testování pouze modulu *PORT_SELECTOR*. Pro ověření správné činnosti i při generování expiračních záznamů bylo testování provedeno se zapnutým generátorem (generátor je využíván pouze při čtení záznamů ze stavové tabulky, pro aktualizace není třeba).

7.3 Testování celého firewallu

Po ověření správné funkčnosti jednotlivých modulů je třeba provést testování kompletního stavového firewallu. Toto testování je provedeno ve dvou krocích. V prvním kroku dojde k simulování stavového firewallu v ModelSimu, druhý krok spočívá v testování za běhu stavového firewallu v FPGA obvodu karty COMBOv2 během reálného provozu.

7.3.1 Simulace

Při simulaci je třeba vytvořit vhodná vstupní data pro co nejlepší otestování pokud možno co nejvíce stavů implementovaného firewallu. Pro vytvoření vstupních dat simulace byl vytvořen nástroj `pcapToHanic`, který provede konverzi obsahu `cap`[31] souborů do formátu vstupních dat simulace. Takto lze např. pomocí programů Wireshark nebo Tcpdump provést zachycení jakékoliv reálné či uměle vytvořené komunikace a záznam použít jako vstupní data simulace. Nástroj lze nalézt na přiloženém CD. Cílem tohoto testování bylo sledování správné reakce celého firewallu na různé druhy vstupních dat.

Vstupní data obsahovala několik různých TCP spojení, která sloužila pro otestování chování při navazování, během existence a při ukončování spojení. Testovací data také obsahovala špatné pořadí TCP paketů či opakování některých paketů (může dojít např. pokud nějaký paket není potvrzen nebo dojde k výpadku paketu při navazování spojení). Došlo i k otestování expirace záznamu v různých stavech TCP komunikace a pro jistotu také k otestování dalších komunikačních protokolů.

Důležitým bodem testování celého firewallu je zaplnění komunikačních front. Kritickým místem je čtení informací ze stavové tabulky, které trvá dva takty. Z tohoto důvodu je omezeno generování příkazů pro expiraci pouze na jeden ze zadaného počtu taktů. Perioda generování těchto příkazů byla vhodně nastavena právě při simulaci stavového firewallu.

7.3.2 Hardware

Testování funkčnosti v hardwaru je obdobné jako simulace. Do paměti pravidel je postupně nahráno několik zkušebních konfigurací a cílem testu je pomocí vygenerovaného provozu sledovat průchod jednotlivých spojení stavovým firewallem podle konfigurace. Ke generování provozu sloužil hardwarový tester Spirent TestCenter[2] se schopností generovat různé množiny paketů s přenosovou rychlostí až 10 GBit/s a pro sledování průchodu jednotlivých paketů nástroj `sze2loopback`.

7.4 Měření výkonnosti

Kromě testů správnosti je třeba také provést měření týkající se propustnosti[1] stavového firewallu. Pro měření výkonnosti byly použity stejné nástroje jako pro testování funkčnosti v podkapitole 7.3.2. Hardwarový tester je schopen generovat pakety s maximální přenosovou rychlostí 10 GBit/s. Tato rychlost tvořila první krok měření propustnosti, další rychlosti byly vypočítány pomocí tzv. binárního půlení (5 Gbit/s, 2.5 GBit/s atd.).

Při měření jsou na oba vstupní porty generovány pakety o velikostech 78 B, 343 B, 789 B, 1144 B a 1500 B. Tyto rychlosti jsou zvoleny s ohledem na možnosti generátoru paketů. Zároveň docházelo ke změně počtu procházejících TCP spojení. Pro účely měření stačí sledovat jen TCP spojení, protože jejich průchod je vlivem spouštění transakcí nad daty ve

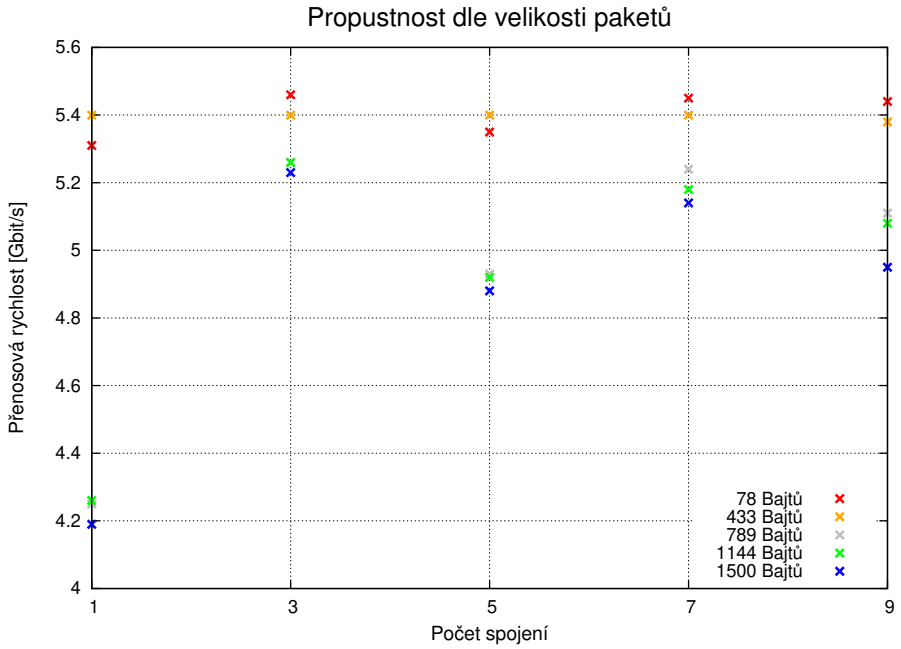
[2] http://www.spirent.com/Solutions-Directory/Spirent-TestCenter/

Počet spojení \ Velikost paketu	78 B	433 B	789 B	1144 B	1500 B
1	5.31	5.40	4.25	4.26	4.19
3	5.46	5.40	5.26	5.26	5.23
5	5.35	5.40	4.93	4.92	4.88
7	5.45	5.40	5.24	5.48	5.14
9	5.44	5.38	5.11	5.08	4.95

Tabulka 7.1: Propustnost stavového firewallu [Gbit/s]

stavové tabulce a čekání na jejich ukončení časově nejnáročnější. Během testování je stavový firewall nastaven tak, že ke generování příkazu pro ověření stáří záznamu dochází jednou za 1220 taktů hodinového signálu (cca 10 μs). Funkčnost generování příkazů je popsána v 5.4.1.

V grafu 7.1 a tabulce 7.1 lze vidět přenosovou rychlost, při které již nedochází ke ztrátě paketů pro různé velikosti paketů a počty spojení.



Obrázek 7.1: Propustnost stavového firewallu

Z měření výkonnosti stavového firewallu vyplývá, že stavový firewall vykazuje nejnižší propustnost pro dlouhé TCP pakety (1500 B) patřící do menšího počtu spojení. Je to způsobeno čekáním na ukončení spuštěných transakcí nad daty ve stavové tabulce, během kterého dochází k přeplnění vstupních bufferů karty. Zároveň je limitujícím faktorem také serializace přístupů ke stavové tabulce. Při rychlosti 10 Gbit/s musí být stavová tabulka schopna pracovat s rychlostí 20 GBit/s. Výsledky měření jsou přiloženy na CD.

Kapitola 8

Závěr

Cílem práce bylo provést návrh a implementaci stavového firewallu na platformě NetCOPE, což je platforma pro rychlý vývoj aplikací běžících na technologii FPGA.

Práce se ve své první částí zabývá popisem platformy NetCOPE. Popsány jsou hlavně části potřebné pro síťovou komunikaci a také části použité v existující bezestavové implementaci firewallu. Dále v této části úvádí do problematiky filtrování paketů pomocí zařízení s názvem firewall. Věnuje se principu paketového filtrování a uvádí jeho nevýhody, které daly podnět ke vzniku stavového filtrování a aplikačnímu filtrování. Podkapitola popisující stavové filtrování slouží zároveň jako analýza požadavků pro stavové filtrování síťové komunikace.

Ve druhé části se věnuje detailnímu návrhu architektury stavového filtru a jeho zavedení do již existující bezestavové implementace. Při návrhu byl kladen důraz na snadnou rozšiřitelnost aplikace co se týče sledování stavu dalších protokolů kromě protokolu TCP. Návrh stavového firewallu byl koncipován tak, aby bylo třeba upravovat co nejméně modulů použitých v bezestavové implementaci. V části popisující implementaci je popsána komunikace mezi jednotlivými moduly a také logika složitějších modulů. Také jsou zde popsány nutné úpravy konfiguračního softwaru.

Třetí část práce obsahuje testování správnosti jednotlivých modulů pomocí simulačního nástroje ModelSIM. Testování funkčnosti celého firewallu je realizováno jak v rámci simulace, tak přímo nahráním stavového firewallu do karty COMBOv2. Stejným způsobem byly prováděny i testy propustnosti firewallu.

Během implementace stavového filtrování se objevila celá řada omezení. Hlavním omezením implementace je stavová tabulka a s ní související identifikace spojení. Stavová tabulka je realizována pamětí, jejíž kapacita je v rámci FPGA obvodu omezená. S tím souvisí maximální šířka sběrnice sloužící pro adresaci této paměti. Další nevýhodou je vyhledávání ve stavové tabulce. Jelikož paměť poskytuje pro každou adresu pouze jednu položku, nelze žádným způsobem zabránit kolizi. Možná řešení tohoto problému jsou uvedena v podkapitole 8.1. Jednoznačná identifikace je hlavním problémem při rozlišování jednotlivých spojení. Použití všech uvažovaných položek komunikace bez jakéhokoliv dalšího zpracování použít nelze. Pokud uvažujeme IPv4 adresaci, tak identifikace pomocí IP adres (2x32 bitů), portů (2x16 bitů) a protokolu (8 bitů), dostaneme jednoznačnou identifikaci o délce 104 bitů, takže paměť by obsahovala 2^{104} položek. Položka stavové tabulky má v současné implementaci 22 bitů, což vede na paměť o velikosti cca $3 * 2^{104}$ bajtů. Rychlá paměť s touto kapacitou v současné době neexistuje.

8.1 Možná rozšíření

Jak již bylo napsáno, hlavním problémem implementace je stavová tabulka, jednoznačná identifikace spojení a zabránění kolizí.

Kolize je možné částečně vyřešit pomocí vícecestné paměti (podobně jako jsou řešeny cache paměti v běžném PC). Kromě adresace konkrétní položky v paměti je třeba také doplnit příznak, pomocí kterého dojde k rozpoznání, ve které cestě paměti se záznam nachází. Tento princip však není úplným řešením, protože může nastat situace, kdy se shoduje vypočítaná adresa a příznak. Zároveň je třeba při vytváření nového záznamu sledovat, která z cest má volné místo k zápisu.

Jiným řešením je použití procesoru a index-sekvenčního přístupu v hashovací tabulce. Tento procesor by obsahoval pomocnou cache paměť, která by obsahovala např. nejčastěji se vyskytující nebo nejnovější záznamy o stavu komunikací. O ostatní nebo nově vytvářené záznamy by se staral procesor, který by měl k dispozici index-sekvenční vyhledávání v hashovací tabulce. Důležité by bylo zajistit konzistenci dat v cache paměti a hashovací tabulce. Při přístupu k záznamu, který není ve stavové tabulce, by došlo k jeho nakopírování do cache, případně nahrazení nejméně používané položky v cache paměti. Procesor by mohl být optimalizován pro práci pouze se stavovou tabulkou a pamětí cache, výpočet identifikace spojení by zajišťoval hardwarový obvod. Tento systém by mohl vyřešit i problém kolizí, protože je možné v položce paměti ukládat také IP adresy, porty a protokol spojení. Takto lze provést porovnání všech identifikátorů komunikace jednotlivě.

Jednoduchým rozšířením současné implementace je možnost sledování stavu protokolů UDP a ICMP. V případě těchto dvou protokolů je komunikace považována za navázanou v případě průchodu prvního povoleného paketu. Rozšíření bude spočívat v přivedení signálů, které jsou nastaveny v případě zkoumání UDP, případně ICMP paketu. Obvod pro vyhodnocení komunikace pak musí označit spojení ihned za existující. Jelikož UDP a ICMP spojení nemají žádný mechanismus pro ukončování, je možné záznam z tabulky odstranit pouze v případě vypršení platnosti.

Složitější je rozšíření o inspekci na aplikační vrstvě. Protokoly jako FTP či H.323 navazují několik spojení na různých portech, přičemž čísla těchto portů se přenášejí ve zprávách při navázání komunikace. Pro tento účel je třeba provést rozšíření modulu HFE, aby prováděl extrakci příslušných položek při detekci FTP či ostatních protokolů. Zároveň je třeba čísla portů uchovávat pro pozdější filtrování navázaných komunikací. Extrakci lze realizovat opět pomocí procesoru, nebo jednoduchého modulu pro každý protokol, který navazuje více spojení bez předem známých portů.

Třetím možným vylepšením stavového firewallu je zvýšení propustnosti. Jedním z možných míst je stavová tabulka, kde by bylo možné urychlit čtení paměti o jeden hodinový takt. Pro odstranění serializace přístupů ke stavové tabulce je možné zavést pro každý port dvě cache paměti obsahující kopie informací o stavu komunikace. Je však třeba zajistit udržování koherence obou pamětí.

Literatura

[1] Bradner, S.; McQuaid, J.: Benchmarking Methodology for Network Interconnect Devices. RFC 2544 (Informational), Březen 1999, updated by RFC 6201.
URL http://www.ietf.org/rfc/rfc2544.txt

[2] CESNET: Projektová dokumentace. Veřejně nedostupná.

[3] Commission, U.: *Certain EPROM, EEPROM, Flash Memory and Flash Microcontroller Semiconductor Devices and Products Containing Same, Inv. 337-TA-395 (Reconsideration)*. DIANE Publishing, apr. 1999, ISBN 978-1-4578-2472-2.
URL http://books.google.cz/books?id=wdaKZSfQ1IMC

[4] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; aj.: *Introduction to algorithms*. MIT Press, třetí vydání, 2001, ISBN 0-262-03293-7, s. 245–249.

[5] Davidson, J.; Peters, J.; Gracely, B.: *Voice Over Ip Fundamentals*. Cisco Press Fundamentals Series, Cisco Press, 2000, ISBN 978-1-5787-0168-1, 229–249 s.
URL http://books.google.cz/books?id=S5P7-Xtq7W8C

[6] Hank, A.: Návrh síťových aplikací na platformě NetCOPE. 2009, diplomová práce.

[7] Honzík, J. M.: *Algoritmy - studijní opora*. FIT, 2007, veřejně nedostupná.

[8] IANA: Service Name and Transport Protocol Port Number Registry. 2012-04-20 [cit. 2012-04-23].
URL http://www.iana.org/assignments/service-names-port-numbers/

[9] Koranda, K.: Přizpůsobení platformy NetCOPE pro kartu NetFPGA. 2011, bakalářská práce.

[10] Martínek, T.; Kořenek, J.: Přednášky předmětu Pokročilé číslicové systémy. Veřejně nedostupné.

[11] Málek, T.; Martínek, T.; Kořenek, J.: GICS: Generic Interconnection System. In *2008 International Conference on Field Programmable Logic and Applications*, IEEE Computer Society, 2008, ISBN 978-1-4244-1960-9, s. 263–268.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8735

[12] Peterson, W.; Brown, D.: Cyclic Codes for Error Detection. *Proceedings of the IRE*, ročník 49, č. 1, jan. 1961: s. 228 –235, ISSN 0096-8390, doi:10.1109/JRPROC.1961.287814.

[13] Pinker, J.: *Číslicové systémy a jazyk VHDL*. Praha: BEN - technická literatura, 2006, ISBN 80-7300-198-5.

[14] Postel, J.: User Datagram Protocol. RFC 768 (Standard), Srpen 1980.
URL http://www.ietf.org/rfc/rfc768.txt

[15] Postel, J.: Internet Control Message Protocol. RFC 792 (Standard), Září 1981, updated by RFCs 950, 4884.
URL http://www.ietf.org/rfc/rfc792.txt

[16] Postel, J.: Transmission Control Protocol. RFC 793 (Standard), Září 1981, updated by RFCs 1122, 3168, 6093.
URL http://www.ietf.org/rfc/rfc793.txt

[17] Postel, J.; Reynolds, J.: File Transfer Protocol. RFC 959 (Standard), Říjen 1985, updated by RFCs 2228, 2640, 2773, 3659, 5797.
URL http://www.ietf.org/rfc/rfc959.txt

[18] Reynolds, J.: Assigned Numbers: RFC 1700 is Replaced by an On-line Database. RFC 3232 (Informational), Leden 2002.
URL http://www.ietf.org/rfc/rfc3232.txt

[19] Scarfone, K.; Mell, P.: Guide to Intrusion Detection and Prevention Systems (IDPS) Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*, ročník 800, č. 94, 2007: str. 94.
URL http://www.ozus.com/NIST/PDF/SP800-94.pdf

[20] Schulzrinne, H.; Rao, A.; Lanphier, R.: Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard), Duben 1998.
URL http://www.ietf.org/rfc/rfc2326.txt

[21] Stephen Northcutt: *Bezpečnost počítačových sítí*. Computer Press, první vydání, 2005, ISBN 80-251-0697-7, 592 s.

[22] Strebe, M.; Perkins, C.: *Firewally a proxy-servery*. Computer Press, první vydání, 2003, ISBN 80-7226-983-6, 450 s.

[23] The Liberouter Project Team: Projektová dokumentace. Veřejně nedostupná.

[24] The Liberouter Project Team: NetCOPE Platform Handbook. 2010-07-14 [cit. 2011-09-28].
URL http://www.liberouter.org/netcope/handbook.html

[25] The Liberouter Project Team: Hashing Network Interface Card Handbook. 2010-10-04 [cit. 2012-05-02].
URL http://www.liberouter.org/hanic/handbook.html

[26] The Liberouter Project Team: Description of COMBO cards. [cit. 2011-09-25].
URL http://www.liberouter.org/hardware.php?flag=2

[27] W3C: Extensible Markup Language (XML). 2012-01-24 [cit. 2012-05-09].
URL http://www.w3.org/XML/

[28] Wikipedia, the free encyclopedia: Tcp state diagram. 2010-12-22 [cit. 2012-04-25].
URL http://en.wikipedia.org/wiki/File:Tcp_state_diagram_fixed_new.svg

[29] Wikipedia, the free encyclopedia: Least significant bit. 2012-01-27 [cit. 2012-04-23].
URL http://en.wikipedia.org/wiki/Least_significant_bit

[30] Wikipedia, the free encyclopedia: Most significant bit. 2012-01-27 [cit. 2012-04-23].
URL http://en.wikipedia.org/wiki/Most_significant_bit

[31] Wireshark project team: Libpcap File Format. 2011-03-30 [cit. 2012-05-09].
URL http://wiki.wireshark.org/Development/LibpcapFileFormat

[32] XILINX: LocalLink Interface Specification [online]. Červen 2005.
URL http://www.xilinx.com/products/intellectual-property/LocalLink_UserInterface.htm

[33] Xilinx: Xilinx Constraints Guide. *Database*, ročník 625, 2009: s. 1–361.
URL http://www.xilinx.com/itp/xilinx10/books/docs/cgd/cgd.pdf

[34] Xilinx: Virtex-5 Family Overview. 2009 [cit. 2011-10-25].
URL http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf

Příloha A

Obsah CD

- /
  - `hanic` ..... Implementace stavového firewallu
    - `doxygen` ..... Dokumentace vygenerovaná nástrojem Doxygen
    - `git` ..... GIT repozitář s provedenými úpravami
    - `mcs` ..... Konfigurační stream pro FPGA a záznamy ze syntézy
    - `README.txt` ..... Informace o provedených úpravách designu
    - `README_HANICCTL.txt` ..... Informace o provedených úpravách konfiguračního nástroje
  - `measurement` ..... Obsahuje výsledky měření propustnosti
  - `test` ..... Složka obsahující nástroje použité pro simulaci
    - `pcapToHanic` ..... Převodník z .cap formátu na vstupní data
    - `sendgenerator` ..... Pomocný skript pro generování vstupů
    - `spirent`. Projekty nástroje Spirent TestCenter použité pro testování a měření
    - `tcprewriter` ..... Skript sloužící pro změnu IP adresy v .cap souborech
  - `text` ..... Zdrojové soubory technické zprávy ve formátu LaTeXvčetně obrázků
  - `xzizka08.pdf` ..... Vysázený text technické zprávy s funkčními odkazy

Příloha B

Využití zdrojů

Rozpis použitých zdrojů v bezestavové i stavové verzi firewallu získaný při syntéze.

B.1 Bezestavový firewall

```
Device Utilization Summary:

   Number of BUFDSs                          3 out of 8      37%
   Number of BUFGs                          17 out of 32     53%
   Number of BUFGCTRLs                       1 out of 32      3%
   Number of CRC64s                          4 out of 16     25%
   Number of DCM_ADVs                        2 out of 12     16%
   Number of DSP48Es                         9 out of 128     7%
   Number of FIFO36_72_EXPs                  6 out of 212     2%
   Number of GTP_DUALs                       8 out of 8     100%
   Number of ILOGICs                         7 out of 800     1%
   Number of External IOBs                 500 out of 640    78%
      Number of LOCed IOBs                 500 out of 500   100%

   Number of IODELAYs                        1 out of 800     1%
   Number of External IPADs                 38 out of 690     5%
      Number of LOCed IPADs                 38 out of 38    100%

   Number of OLOGICs                         9 out of 800     1%
   Number of External OPADs                 32 out of 32    100%
      Number of LOCed OPADs                 32 out of 32    100%

   Number of PCIEs                           1 out of 1     100%
   Number of PLL_ADVs                        3 out of 6      50%
   Number of RAMB18X2s                       2 out of 212     1%
   Number of RAMB18X2SDPs                   41 out of 212    19%
   Number of RAMB36SDP_EXPs                  5 out of 212     2%
   Number of RAMB36_EXPs                    80 out of 212    37%
   Number of SYSMONs                         1 out of 1     100%
   Number of Slices                      17176 out of 24320  70%
   Number of Slice Registers             33906 out of 97280  34%
      Number used as Flip Flops          33890
      Number used as Latches                16
      Number used as LatchThrus              0

   Number of Slice LUTS                  40187 out of 97280  41%
   Number of Slice LUT-Flip Flop pairs   51697 out of 97280  53%
```

B.2 Stavový firewall

Device Utilization Summary:

Number of BUFDSs	3	out of 8	37%
Number of BUFGs	17	out of 32	53%
Number of BUFGCTRLs	1	out of 32	3%
Number of CRC64s	4	out of 16	25%
Number of DCM_ADVs	2	out of 12	16%
Number of DSP48Es	9	out of 128	7%
Number of FIFO36_72_EXPs	6	out of 212	2%
Number of GTP_DUALs	8	out of 8	100%
Number of ILOGICs	7	out of 800	1%
Number of External IOBs	500	out of 640	78%
Number of LOCed IOBs	500	out of 500	100%
Number of IODELAYs	1	out of 800	1%
Number of External IPADs	38	out of 690	5%
Number of LOCed IPADs	38	out of 38	100%
Number of OLOGICs	9	out of 800	1%
Number of External OPADs	32	out of 32	100%
Number of LOCed OPADs	32	out of 32	100%
Number of PCIEs	1	out of 1	100%
Number of PLL_ADVs	3	out of 6	50%
Number of RAMB18X2s	3	out of 212	1%
Number of RAMB18X2SDPs	38	out of 212	17%
Number of RAMB36SDP_EXPs	5	out of 212	2%
Number of RAMB36_EXPs	81	out of 212	38%
Number of SYSMONs	1	out of 1	100%
Number of Slices	17827	out of 24320	73%
Number of Slice Registers	34335	out of 97280	35%
Number used as Flip Flops	34319		
Number used as Latches	16		
Number used as LatchThrus	0		
Number of Slice LUTS	42713	out of 97280	43%
Number of Slice LUT-Flip Flop pairs	53787	out of 97280	55%

Příloha C

Konfigrační soubor s pravidly

Vzorový konfigurační soubor pro demonstraci konfigurace stavového filtrování:

```
<config id="0">
<sets>
  <set id="1">
    <IPS>
      <data>192.168.22.5</data> <mask>255.255.255.255</mask>
    </IPS>
    <IPD>
      <data>172.16.240.3</data> <mask>255.255.255.255</mask>
    </IPD>
    <PORTS>80</PORTS> <PORTD>80</PORTD> <PROTO>6</PROTO>
  </set>
</sets>
<rules>
  <rule priority="1">
    if (IPS in 1) and (IPD in 1) and (PORTD in 1) and (PROTO in 1)
      then allow 0/0 UPDATE
  </rule>
  <rule priority="1000">if always then deny</rule>
</rules>
</config>

<config id="1">
<sets>
  <set id="1">
    <IPS>
      <data>172.16.240.3</data> <mask>255.255.255.255</mask>
    </IPS>
    <IPD>
      <data>192.168.22.5</data> <mask>255.255.255.255</mask>
    </IPD>
    <PORTS>80</PORTS> <PORTD>80</PORTD> <PROTO>6</PROTO>
  </set>
</sets>
<rules>
  <rule priority="1">
    if (IPS in 1) and (IPD in 1) and (PORTS in 1) and (PROTO in 1)
      then allow 0/0 CHECK
  </rule>
  <rule priority="1000">if always then deny</rule>
</rules>
</config>
```

Jelikož původní implementace obsahovala konfiguraci každého vstupního rozhraní odděleně, je třeba na toto dát pozor i při konfiguraci stavového filtrování.

Konfigurační soubor obsahuje ukázku stavového filtrování HTTP (port 80) komunikace, kterou vždy inicializuje stroj ve vnitřní síti s IP adresou 192.168.22.5 na stroj s IP adresou 172.16.240.3 v jiné síti. Port, na kterém se nachází vnitřní síť, tak musí být konfigurován se stavovou akcí UPDATE, druhý port musí obsahovat akci CHECK. Ostatní komunikace jsou zakázané. V případě použití akce UPDATE nebo CHECK na jiný protokol než TCP skončí nahrávání pravidel úspěšně, ale firewall bude tyto akce ignorovat.