UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tomo Ceferin

MODULARNA PROGRAMSKA REŠITEV MOSTA ZA PROTOKOL ETHERNET

MAGISTRSKO DELO

Mentor: prof. dr. Nikolaj Zimic

Ljubljana, 2011

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.: 143-MAG-RI/2011
Datum: 19. 09. 2011

Tomo CEFERIN, univ. dipl. inž. rač. in inf.

L j u b l j a n a

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Modularna programska rešitev mosta za protokol Ethernet**

Adaptable Ethernet bridge

Tematika naloge:
Omrežni mostovi za protokol Ethernet so zelo raznolike naprave. Poleg osnovne naloge povezovanja omrežnih segmentov, lahko opravljajo še množico drugih nalog. Iz nekdaj preprostih naprav so se omrežni mostovi razvili v zapletene sistemske rešitve, to pa vpliva tudi na način njihovega načrtovanja, razvoja in vzdrževanja. Eno močnejših gonil njihovega razvoja je uporaba v velikih omrežjih ponudnikov telekomunikacijskih storitev, kjer nadomeščajo nekdaj prevladujočo tehnologijo ATM. To omrežnim mostovom nalaga dodatne naloge, katerih širitvi ni videti konca.

V magistrski nalogi prikažite širino funkcionalnosti, ki jih pričakujemo pri sodobnih omrežnih mostovih namenjenih vgradnji v okolje ponudnikov telekomunikacijskih storitev in analizirajte operacije, ki jih tak most opravlja. Primerjajte različne rešitve in predlagajte izboljšave, ki naj poenostavijo razvoj, odkrivanje napak in vzdrževanje.

M e n t o r:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

D e k a n:

prof. dr. Nikolaj Zimic

Original izdane teme

I Z J A V A O A V T O R S T V U

magistrskega dela

Spodaj podpisani Tomo Ceferin z vpisno številko 24930470 sem avtor magistrskega dela z naslovom *Modularna programska rešitev mosta za protokol Ethernet*.

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod vodstvom mentorja prof. dr. Nikolaja Zimica;
- so elektronska oblika magistrskega dela, naslova (slov., angl.) in povzetka (slov., angl.) ter ključne besede (slov., angl.) identične s tiskano obliko magistrskega dela in
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 3.10.2011. Podpis avtorja:

1 Zahvala

Tega dela ne bi bilo, če ne bi bil deležen pomoči številnih ljudi. Nekateri so mi pomagali neposredno, drugi predvsem s potrpežljivostjo med mojo odsotnostjo.

Med prvimi gre zahvala mentorju, prof. dr. Nikolaju Zimicu. Njemu gre zasluga, da sem začeto tudi končal, sicer bi naloga najbrž neslavno končala.

Da veliko časa namenjam drugim stvarem so najbolj opazili Jana in starši, a so to pogumno prenašali.

Sodelavca Jože Čebašek in Jure Buble sta bila pogosto prva prisiljena prisluhniti mojim idejam.

Brez pomoči Jane Zagožen bi se le težko dokopal do nekaterih IEEE člankov, za kar se ji na tem mestu iskreno zahvaljujem.

2 Kazalo

3 Kratice, okrajšave, simboli

ACL	Access Control List
A/D	Analog to Digital
ADSL	Asymmetric DSL
AG	Aggregation Node
AN	Access Node
ARL	Application Rate Limiting
ATM	Asynchronous Transfer Mode
BER	Bit Error Rate
BGP	Border Gateway Protocol
BPDU	Bridge Protocol Data Unit
BRAS	Broadband Remote Access Server
CFI	Canonical Format Indicator
CFM	Connectivity Fault Management
CPE	Customer Premisses Equipment
CPU	Central Processing Unit
D/A	Digital to Analog
DEI	Drop Eligible Indicator
DLF	Destination Look-up Failure
DMAC	Destination MAC address
DNS	Domain Name System
DPI	Deep Packet Inspection
DS	Down-Stream
DSCP	Differentiated Services Code Point
DSL	Digital Subscriber Line
DSLAM	Digital Subscriber Line Access Multiplexer
EAPS	Ethernet Automatic Protection Switching
ECN	Explicit Congestion Notification
EFM	Ethernet Fault Management
ERPS	Ethernet Ring Protection Switching
ES	Ethernet Switch
FCS	Frame Check Sequence

FTP	File Transfer Protocol
HDTV	High Definition TV
HTTP	Hyper-Text Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ICMPv4	ICMP version 4
ICMPv6	ICMP version 6
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Multicast Protocol
IHL	Internet Header Length
IP	Internet Protocol
IPSec	IP Security
ISS	Internal Sublayer Service ([5])
HTTP	Hyper-Text Transport Protocol
HTTPS	Secure HTTP
L2	OSI Level 2 – Link Layer
L3	OSI Level 3 – Network Layer
L4	OSI Level 4 – Transport Layer
LAN	Local Area Network
LSB	Least Significant Bit
MAC	Media Access Control
MSB	Most Significant Bit
MSTP	Multiple STP
NAT	Network Address Translation
NDA	Non-Disclosure Agreement
OAM	Operations, Administration and Management
OSPF	Open Shortest Path First
OUI	Organizationally Unique Identifier
PCR	Peak Cell Rate
PDU	Protocol Data Unit
PHY	Physical interface
PPP	Point-to-Point Protocol
PPPoE	PPP-over-Ethernet

PPPoED	PPPoE Discovery
PPPoES	PPPoE Session
QoS	Quality of Service
RIP	Routing Information Protocol
RSTP	Rapid STP
RTP	Real-Time Transport Protocol
SMAC	Source MAC address
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SCR	Sustained Cell Rate
SLA	Service Level Agreement
SoC	System-on-a-Chip
SOHO	Small Office, Home Office
SSH	Secure Shell
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
T/L	Type/Length
TLS	Transport Layer Security
TLV	Type-Length-Value
ToS	Type of Service
TTL	Time-To-Live
US	Up-Stream
VLAN	Virtual LAN
VoIP	Voice-over-IP
VPN	Virtual Private Network
XDR	External Data Representation

4 Povzetek

Pripravili smo pregled elementov, ki sestavljajo računalniška omrežja, zgrajena z uporabo protokola Ethernet. V nadaljevanju smo pripravili kratek pregled nekaterih pogosto uporabljenih omrežnih protokolov. Izbrali smo protokole, s katerimi je običajen uporabnik v stiku vsakodnevno – večina nevede.

Sledi predstavitev mostov protokola Ethernet in funkcionalnih sklopov, ki predstavljajo osnovo mostov družine DSLAM. Ta se od običajnih mostov loči po večjem številu funkcij, uporablja pa se v pristopnih vozliščih, na meji med omrežjem ponudnikov telekomunikacijskih storitev na eni ter domačimi in poslovnimi uporabniki na drugi strani.

Tri različne izvedbe mostov smo primerjali glede na naslednje lastnosti:

- spremembe – enostavnost dodajanja sprememb in vzdrževanja;
- odkrivanje napak – možnost odkrivanja napak v izvedbi in nastavitvah;
- podvajanje – isti sklopi, ki so izvedeni na več mestih, po možnosti celo na različne načine;
- hitrost – hitrost delovanja;
- upravljanje – enostavnost upravljanja.

Ob pomoči primerjave smo pripravili seznam lastnosti, ki jih od mosta pričakujemo, da bi bilo njegovo vzdrževanje čim lažje. Pripravili smo koncept izboljšane arhitekture modularnega omrežnega mosta.

Konceptualni predlog smo nadgradili s predlogom za izvedbo v okolju jedra operacijskega sistema Linux in s konkretno izvedbo nekaterih funkcionalnosti v predlaganem okolju.

5 Abstract

We prepared an overview of elements that comprise computer networks built using Ethernet protocol. We followed that with an overview of some frequently used network protocols. Our focus was on protocols daily encountered by regular users – mostly not knowingly.

We then present Ethernet bridges and functions that form the basis of DSLAM bridge family. These bridges are separated from ordinary bridges by larger number of functions. They are used as access nodes, occupying the border between service providers and home and business users.

We compared three bridge implementations on the basis of the following:

- changes – ease of modification and maintenance;
- problem discovery – ease of implementation and configuration problem discovery;
- duplication – same functions implemented multiple times, possibly even in different ways;
- speed – speed of operation;
- management – ease of management.

Based on comparison we prepared a list of properties that would make maintenance simpler. We prepared a concept of improved architecture of modular bridge.

We provide a mapping of the abstract concept to Linux kernel and an actual implementation for Linux kernel.

6 Uvod

Danes je zelo veliko računalnikov povezanih v računalniška omrežja. Ko pravimo, da *„imamo internet doma“,* to v resnici pomeni, da je naš domači računalnik povezan v veliko omrežje z mnogimi drugimi računalniki. Kadar ni težav, je s tem običajno konec našega zanimanja za računalniška omrežja.

Povezava dveh računalnikov je preprosta – lahko ju povežemo neposredno z ustreznim povezovalnim kablom, povezava treh ali več pa že zahteva posebne vmesne naprave. Nekatere omrežne tehnologije neposredne povezave več naprav ne dopuščajo, danes zelo razširjen protokol Ethernet pa jih.

Prve povezovalne naprave v omrežjih protokola Ethernet so bili spojniki. Ti omogočajo širitev omrežja na fizičnem nivoju, na nivoju električnih povezav. Gre za poceni rešitev, ki pa prinaša tudi nekatere težave. Te izhajajo iz dejstva, da je gre v bistvu za izpeljanko protokola ALOHA.

Težave spojnikov je rešila naslednja generacija naprav – stikala oziroma mostovi. Ti omogočajo veliko boljši izkoristek omrežja. Drugače od spojnikov tudi ne delujejo na fizičnem nivoju, ampak nivo višje, na nivoju povezave. Uvedba mostov je omogočila rast omrežij, ta pa je bila razlog za vedno nove zahteve. Razvoju mostov tako tudi danes, po več kot treh desetletjih, ni videti konca.

Dejstvo, da so omrežni mostovi prisotni skoraj povsod, kjer srečamo računalnike, obenem pa se jih le redko zavedamo, govori samo zase. Kdor ima računalnik, se tega zaveda. Da pa ima danes večina domov tudi en ali dva omrežna mostova, pa ne.

V Sloveniji smo vajeni, da večina računalniške opreme prihaja iz tujine. V primeru mostov ni tako. Podjetje Iskratel iz Kranja v sodelovanju s skupino podizvajalcev že nekaj let razvija in prodaja omrežne mostove, ki pripadajo posebni družini, znani pod kratico DSLAM. Ti predstavljajo pomemben člen v slovenskem širokopasovnem omrežju.

Malo domačih uporabnikov se zaveda, da so naprave, na katere je priključen njihov domači modem, plod slovenskega razvoja. Del domačega razvoja je tudi v nekaterih modemih. Razvoj omrežnih mostov je za nas zato tudi praktičnega pomena. Iz istega razloga nas zanima tudi, kako narediti razvoj bolj predvidljiv, lažji, če je mogoče, pa bi se mu radi tudi izognili in tako prihranili čas.

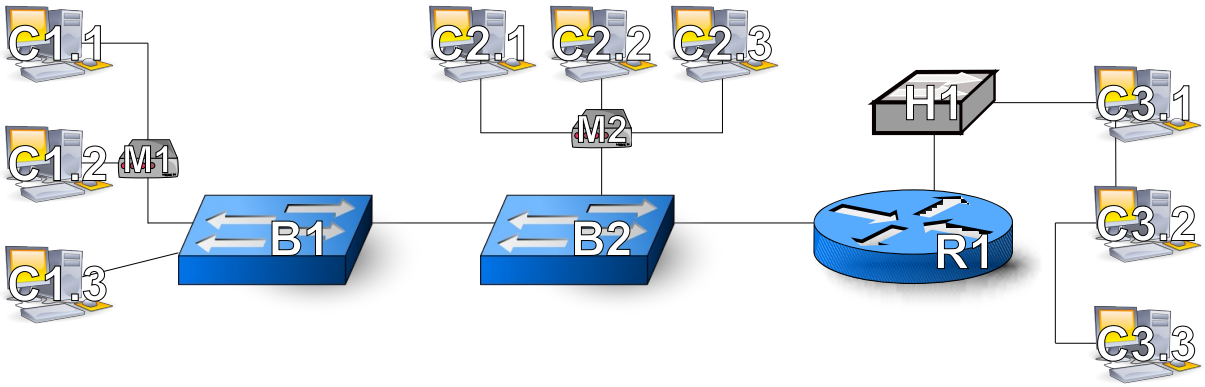
V naslednjem poglavju bomo pregledali nivoje omrežij, omrežno opremo in omrežne protokole. Sledi poglavje, posvečeno mostovom in funkcijam, ki jih morajo opravljati.

V poglavju 9 bomo našteli lastnosti, ki jih pričakujemo od prilagodljivega modularnega mosta, predstavili njegovo arhitekturo in podali predlog preslikave v okolje operacijskega sistema Linux. Sledeče poglavje je namenjeno primerom uporabe, kjer bomo osnovne akcije in pogoje predlaganega mosta uporabili za izvedbo različnih funkcionalnosti, predstavljenih v prejšnjih poglavjih.

V poglavju 11 bomo predstavili še konkretno izvedbo modularnega mosta v okolju operacijskega sistema Linux.

7 Računalniška omrežja

Računalniška omrežja so prisotna vsepovsod, vajeni pa smo, da večinoma delujejo brez težav. Nekaj več razmisleka jim posvetimo šele, ko ne delujejo, kot od njih pričakujemo.



Slika 1: Izsek iz računalniškega omrežja.

Za boljše, predvsem pa pravilno razumevanje težav, ki lahko nastopijo, in izzivov, s katerimi se ukvarjajo načrtovalci in vzdrževalci računalniških omrežij, poglejmo, kakšno je sodobno računalniško omrežje, preko katerega se povezujejo naši računalniki (slika 1).

Na sliki vidimo gruče računalnikov (C1.1-C3.3), povezujejo pa jih modemi (M1-M2), spojniki (H1), mostovi (B1-B2) in usmerjevalniki (R1).

Najprej bomo razložili, zakaj in kako omrežja delimo na nivoje. Sledila bo predstavitev na nivoju fizične opreme. Prikazali bomo oprijemljive elemente, ki sestavljajo dele računalniškega omrežja, in predstavili odnose med njimi. Kasneje bomo temu dodali še nekaj protokolov in razložili, kako vse skupaj pripelje do delujočega omrežja.

7.1 Nivoji omrežja

Računalniška omrežja povezujejo računalnike. Ti so lahko zelo različni. Na eni strani imamo superračunalnike, na drugi pa vgrajene sisteme, ki so narejeni na osnovi računalnikov na enem čipu (angl. *System-on-a-Chip*, SoC). Pri takem razponu naprav se srečamo z raznovrstnimi razlikami. Manjši sistemi so 8-bitni, največji 64- ali celo 128-bitni. Nekateri uporabljajo način zapisa „majhnega konca“ (angl. *little endian*), drugi „velikega konca“ (angl. *big endian*)[1]. Velike so tudi razlike v hitrosti računalnikov, količini pomnilnika, storitvah, ki jih ponujajo, itd.

Seveda bi bilo mogoče zasnovati omrežje kot enotno komponento, ki bi premoščala vse naštete razlike, a tak pristop bi bil neroden, saj bi močno otežil razvoj in vzdrževanje

1 Pravilo se nanaša na to, kako procesor zapiše podatke, ki so večji od enega zloga, v pomnilnik. Pri načinu „majhnega konca“ je na nižjem naslovu manj pomemben zlog, pri „velikem koncu“ pa najbolj pomemben zlog.

komponente. Poleg tega bi bil velik del funkcionalnosti uporabljen le na redkih sistemih, na vseh ostalih pa bi po nepotrebnem zasedal prostor.

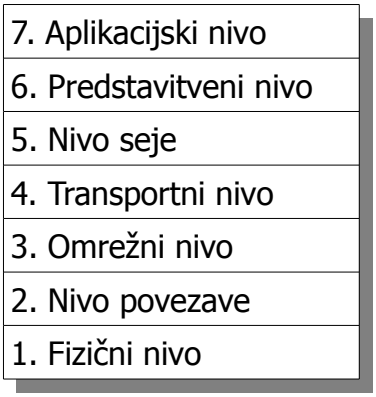
Snovalci omrežij so uporabili pristop *deli in vladaj* – problem izgradnje omrežja so razdelili na podprobleme, potem pa se spopadli ločeno z vsakim od njih:

- fizična povezava računalnikov,
- dogovarjanje med udeleženci,
- pošiljanje posameznih paketov,
- pošiljanje tokov,
- zapis podatkov na način, ki ga razumejo vsi udeleženci,
- predstavitev podatkov.

Rezultat je delitev omrežja na nivoje, od katerih vsak prevzame določen del celotne naloge. Za začetek potrebujemo dogovor o nivojih in nalogah, ki jih opravlja vsak od njih.

7.1.1 Referenčni model OSI

Model OSI deli omrežje na nivoje, vsakemu od njih pa pripisuje določene naloge. Gre za referenčni model, prikazan na sliki 2, ki se uporablja predvsem za lažje opisovanje delov omrežij.



Slika 2: Nivoji po modelu OSI [2].

Končna delitev je bila seveda rezultat pogajanj in dogovorov. Kasneje bomo videli, da so se snovalci več omrežnih protokolov odločili združiti funkcionalnosti več nivojev.

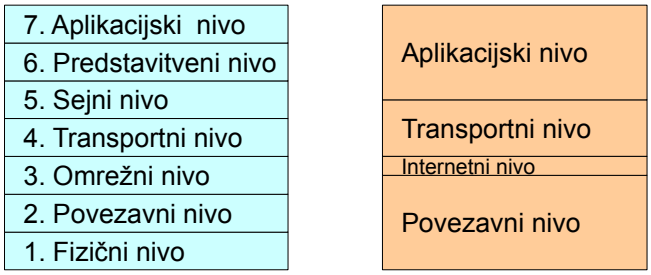
Pri izvedbi posameznih omrežnih protokolov večkrat opazimo, da se začrtanih meja ne držijo povsem natančno. Slika 3 prikazuje primerjavo nivojev modela OSI in protokolnega sklada TCP/IP[2]. Načrtovalci so delitev na preveč nivojev označili za škodljivo ([15]).

Omrežne protokole bomo spoznavali od spodaj navzgor glede na model OSI. Ker smo ga doslej že večkrat omenili, je prav, da ga tudi dokončno predstavimo.

Na nivoje se pogosto sklicujemo, zato je v uporabi skrajšana oblika njihovega označevanja: črki *„L“* dodamo številko nivoja. Drugi nivo tako označimo kot L2.

2 Če smo prvega omenili kot referenčnega za opisovanje nivojev, pa si drugi zasluži naziv referenčne izvedbe omrežnega sklada.

Za nas so pomembni predvsem spodnji nivoji, zato jih bolj podrobno poglejmo.



Slika 3: Primerjava nivojev modela OSI in protokolnega sklada TCP/IP.

7.1.1.1 Fizični nivo (L1)

Gre za nivo fizičnih povezav. Na tem nivoju rešujemo težave prenosa signala, kodiranja podatkov, odkrivanja partnerjev na nivoju povezave in njihovih zmožnosti. Očitno torej znotraj fizičnega nivoja obstajajo dodatni nivoji, ki rešujejo posamezne probleme.

Zaradi hitrosti je vse implementirano v strojni opremi[3].

7.1.1.2 Nivo povezave (L2)

Obsega nivo omrežnega segmenta, funkcionalno pa omogoča pošiljanje okvirjev[4] med napravami, ki se nahajajo na istem omrežnem segmentu. Poslane okvirje zaznajo vse povezane naprave, vendar pa protokoli omogočajo naslavljanje posameznih med njimi, skupine ali vseh naenkrat. Deli implementacije so običajno še v strojni opremi, vedno več pa tudi v programski.

Danes je najbolj razširjen na nivoju povezave protokol Ethernet[5], v zatonu pa so protokoli ATM, Frame Relay, Token Ring in IPX.

7.1.1.3 Mrežni nivo (L3)

Mrežni nivo je prvi, katerega domet je celotno omrežje. Na tem nivoju je glavna naloga učinkovito usmerjanje paketov[6] proti cilju.

Najbolj razširjen protokol mrežnega nivoja je protokol IP – danes še različica 4 (IPv4), kmalu pa ga bo najprej dopolnila, potem pa nadomestila različica 6 (IPv6). Poleg tega sodijo na tretji nivo tudi protokoli IPSec, ICMP, IGMP in OSPF.

Meje med nivoji zares niso jasne. Protokola ICMP in IGMP oba delujeta nad protokolom IP. Njuna naloga je posredovanje dogodkov in napak v omrežju (protokol ICMP) in podpora razpošiljanju na več ciljnih naslovov (protokol IGMP) – vplivata torej na način usmerjanja ostalih paketov. Zaradi tega ju uvrščamo v tretji nivo.

---

3 Načeloma od strojne opreme potrebujemo le D/A- in A/D-pretvornik, pa bi lahko vse to izvedli tudi programsko. A pri hitrostih, ki skoraj nikoli niso pod 10 Mbit/s, bi to lahko dosegli le s hitrimi splošno namenskimi računalniki, njihov izkoristek pa bi bil slab.

4 Glede poimenovanja enot za izmenjavo obstajajo med protokoli razlike. Protokol Ethernet govori o okvirjih (angl. *frame*), protokol ATM o celicah (angl. *cell*) itd.

5 V resnici protokol Ethernet ni čisti L2-protokol, ampak združuje L1 in L2.

6 Izraz paket (angl. *packet*) uporablja protokol IP.

7.1.1.4 Transportni nivo (L4)

Naloga transportnega nivoja je prenos podatkov med končnimi točkami, ob tem pa skriva razne fizične in logične lastnosti spodnjih nivojev.

Najbolj znana protokola tega nivoja sta TCP in UDP.

7.1.1.5 Sejni nivo (L5)

Naloge sejnega nivoja so vzpostavitev, vzdrževanje in rušenje seje. To vključuje pravilno razvrščanje paketov (teh ne sprejmemo nujno v enakem vrstnem redu, kot so bili odposlani) in zagotavljanje dostave (ob izgubi zahtevamo ponoven prenos).

V protokolnem skladu Internet to nalogo opravljata protokola TCP in RTP.

7.1.1.6 Predstavitveni nivo (L6)

Naloga predstavitvenega nivoja je povezovanje aplikacij. Ukvarja se predvsem s predstavitvijo in kodiranjem podatkov. Predstavitveni nivo recimo rešuje težavo različnega načina zapisa števil na različnih procesorjih (gre za tako imenovana načina zapisa *Big* in *Little endian*). Prav tako na predstavitveni nivo sodi enkripcija.

Predstavniki protokolov tega nivoja so SSL, TLS in XDR.

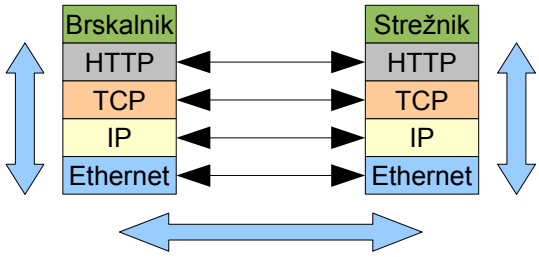
7.1.1.7 Aplikacijski nivo (L7)

Aplikacijski nivo je najbližje uporabniku. Sodeluje z aplikacijami, ki podatke predstavljajo in prevzemajo od uporabnikov in drugih aplikacij.

Primeri protokolov tega nivoja so HTTP, FTP, SMTP in SNMP.

7.1.2 Protokolni sklad

Z izrazom „protokolni sklad“ (angl. *protocol stack*) imamo v mislih skupino protokolov na različnih nivojih, ki sodelujejo pri izvedbi kakšne naloge.

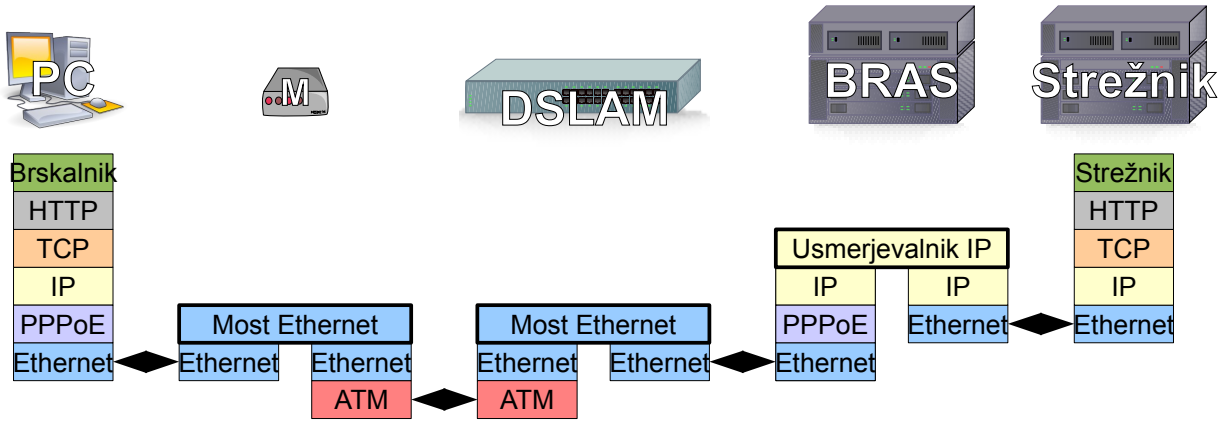


Slika 4: Protokolni sklad HTTP povezave.

Na sliki 4 je prikazan primer delovanja protokolnega sklada v primeru povezave HTTP. Brskalnik preko protokola HTTP zahteva stran od strežnika. Protokol HTTP od nižjega nivoja, to je protokola TCP, zahteva odprtje povezave do strežnika na drugem računalniku. Protokol TCP najprej vzpostavi povezavo, kasneje pa podatkovne bloke, ki jih pripravi protokol HTTP, razdeli v pakete. Te preda protokolu IP, ki jih dostavi na pravi ciljni računalnik. Protokol IP uporabi protokol nivoja 2 Ethernet, da dostavi podatkovne okvirje

sosednjemu računalniku, na katerem teče strežnik. Na enak način teče komunikacija tudi v nasprotni smeri.

Globina protokolnega sklada se lahko tudi spreminja. Na sliki 5 imamo precej bolj realen primer. Prikazuje dogajanje, ko se od doma povežemo preko modema ADSL na nek bližnji spletni strežnik.



Slika 5: Raznoliki mrežni skladi, ki sodelujejo pri preprosti povezavi.

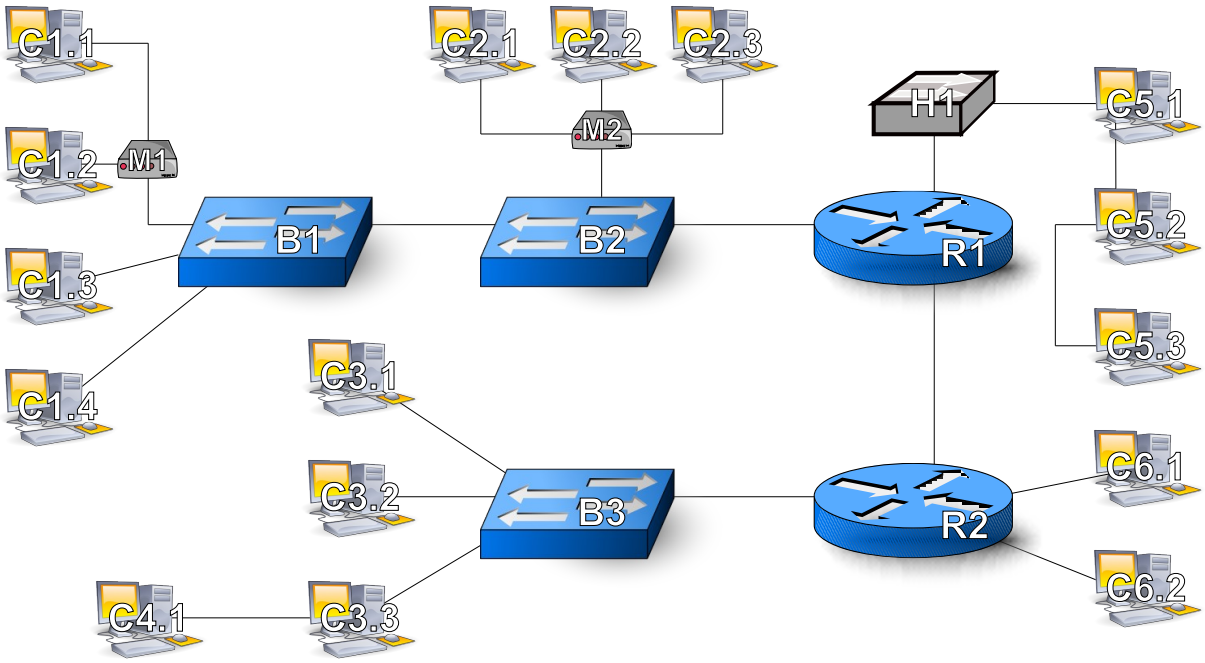
Večina povezav na fizičnem nivoju poteka preko omrežij protokola Ethernet, povezava med domačim modemom in mostom DSLAM (pogovorno velikokrat slišimo izraz „centrala“) pa poteka preko omrežja ATM. A to dejstvo je skrito pred vsemi udeleženci, zavedata se ga le modem in most DSLAM. Prav tako se dejstva, da so podatki vsebovani znotraj seje PPPoE, zavedata le osebni računalnik in usmerjevalnik BRAS (angl. *Broadband Remote Access Server*).

7.2 Omrežna oprema

Oglejmo si fizične gradnike, ki sestavljajo omrežje. Pri tem bomo uporabljali posplošeno sliko omrežja s slike 6, ki vsebuje:

- zaključne točke (računalnike) C1.1–C6.2,
- omrežne segmente vrste „točka-točka“, ki jih sestavljajo povezave računalnik C1.1–modem M1, računalnik C1.2–modem M1, modem M1–most B1, računalnik C1.3–most B1, most B1–most B2, most B2–usmerjevalnik R1, računalnik C5.1–spojnik H1–usmerjevalnik R1[7] itd.
- modema M1-M2,
- spojnik (angl. *hub*) H1,
- mostove B1–B3 in
- usmerjevalnika R1-R2.

7 Kot vidimo, povezava poljubnih dveh naprav predstavlja en omrežni segment. Izjema pri tem so spojniki. Vse naprave, ki so priključene na isti spojnik, spadajo v isti omrežni segment.



Slika 6: Raznoliko omrežje.

7.2.1 Zaključne točke

Zaključno točko (angl. *terminal* ali *terminal equipment*) v omrežju predstavljajo računalniki. Pri tem imamo v mislih računalnike v širšem pomenu besede. Laični uporabniki na televizijski komunikator (angl. *set top box*) ne gledajo kot na računalnik, pa to seveda je. Nič drugače ni s telefonom IP in brezžičnim usmerjevalnikom. Če dodamo še mrežne tiskalnike in omrežno priklopljeni pomnilnik (angl. *Network Attached Storage*, NAS) smo pokrili velik del opreme, ki se priklaplja na računalniška omrežja.

Zaključne točke so izvor in ponor večine prometa v omrežjih. Namen omrežja je prenašanje podatkov med zaključnimi točkami. Na sliki 6 so zaključne točke označene s C1.1–C6.2.

7.2.2 Omrežni segmenti

Omrežni segment (angl. *network segment*) je najmanjši del omrežja. Sestavljajo ga povezave med zaključnimi točkami ter med zaključnimi točkami in mrežno opremo. Povezave so lahko neposredne ali preko povezovalne opreme, pri tem pa je dejavnost povezovalne opreme omejena izključno na izboljšavo električnih ali optičnih lastnosti signala. Primera povezovalne opreme sta spojnik (angl. *hub*) in obnavljalnik (angl. *repeater*).

Naloga omrežnih segmentov je, da med svojimi mejnimi točkami kar se da kvalitetno prenašajo električne ali optične signale.

Na sliki 6 vidimo veliko število omrežnih segmentov. Omrežne segmente predstavljajo naslednje povezave:

- računalnik C1.1 in modem M1;
- računalnik C1.3 in most B1;

- računalnik C4.1 in računalnik C3.3;
- računalnik C5.1, spojnik H1 in usmerjevalnik R1;
- most B1 in most B2;
- most B3 in usmerjevalnik R2;
- usmerjevalnik R1 in usmerjevalnik R2.

Na sliki je seveda še precej več omrežnih segmentov, a za ilustracijo zadostujejo našteti.

7.2.2.1 Omrežni segment in zmogljivost omrežja

Ker omrežni segment združuje naprave na fizičnem nivoju, morajo biti vse vanj povezane naprave glede obnašanja na fizičnem nivoju usklajene. Omrežja protokola Ethernet omogočajo napravam, da se, ob priključitvi v fazi pogajanj (angl. *negotiation*), z drugimi napravami dogovorijo za način delovanja[8].

Če je v omrežni segment povezanih več naprav, potem se bodo povezale v najboljšem načinu, ki ga podpirajo vse. Preprosteje – vsi se bodo prilagodili najmanj sposobni napravi.

To seveda pomeni, da s številom naprav, povezanih v omrežni segment, raste tudi možnost, da naprave ne bodo delovale v najboljšem možnem načinu.

7.2.2.2 Omrežni segment in prepustnost omrežja

Današnja omrežja so večinoma zgrajena tako, da ima vsak segment samo dve zaključni točki. Čeprav vsa tehnološka podlaga omogoča priključitev večjega števila naprav v isti omrežni segment, tega večinoma ne uporabljamo več.

Protokol Ethernet je v svoji osnovi še vedno protokol ALOHA [1]: vse povezane naprave lahko oddajajo naenkrat, če ugotovijo, da je prišlo do trka, pa vsaka od naprav počaka nekaj časa, potem pa poskusi znova. Posledica tega je, da veliko število povezanih naprav poveča možnost trkov in tako zmanjša delež koristno uporabljene prenosne kapacitete. To lastnost so opazili in izmerili že prvi raziskovalci [26][27][28].

Kadar sta v omrežni segment povezani le dve zaključni točki (manj seveda ni smiselno), te nevarnosti ni[9], to pa dopušča boljšo izrabo povezave.

7.2.3 Stikala in mostovi

Stikala ali mostovi so elementi omrežij, ki povezujejo[10] omrežne segmente. To je njihova najbolj osnovna naloga. V večjih omrežjih stikala opravljajo še veliko drugih nalog. Podrobno jih na tem mestu ne bomo opisovali, saj so predmet tega dela in se jim bomo podrobneje posvetili kasneje.

Naloga stikal je, da iz signala, ki ga prejmejo preko mrežnega segmenta, razberejo okvirje (angl. *frame*) in te okvirje posredujejo ustreznim nadaljnjim omrežnim segmentom.

---

8 Postopek samodejnih pogajanj lahko nadomestimo tudi z zahtevanim načinom povezovanja, ki pa lahko tudi ne uspe.

9 To je res, če obe zaključni točki delujeta v polnem dupleksnem načinu (angl. *full duplex*). V tem primeru lahko obe hkrati pošiljata in sprejemata.

10 Stikalo omrežne segmente združuje, segmenti pa še vedno ostanejo ločeni. Spojnik več omrežnih segmentov združi v enega samega.

Za razumevanje delovanja omrežij je pomembno, da stikala delujejo na *drugem nivoju*, na kratko na L2. Pri tem imamo v mislih, da ne delujejo na nivoju neposredne električne (po novem tudi optične) povezave, pač pa se ukvarjajo s podatkovnimi okvirji. Svet stikala predstavljajo neposredno dosegljivi terminali in druga omrežna oprema drugega nivoja.

Če pogledamo na sliko 6 in se osredotočimo na stikalo B1, njegovo okolico sestavljajo M1, C1.3-4. in B2. Stikalo B1 se ne zaveda usmerjevalnika R1, računalnikov C1.1-2 in C2.1-3, stikala B3 itd. Ti elementi za stikalo B1 obstajajo samo posredno: na podlagi opazovanja prometa lahko domnevamo, da za omrežno opremo v njegovi neposredni okolici obstaja še druga.

Slika 7: Poenostavljena zgradba Ethernet okvirja.

Danes najbolj pogost protokol drugega omrežnega nivoja je protokol Ethernet. Slika 7 prikazuje zgradbo okvirja Ethernet. Ta hip sta za nas pomembna dva elementa tega okvirja: ciljni naslov (angl. *destination MAC*, DMAC) in izvorni naslov (angl. *source MAC*, SMAC). Na podlagi teh dveh podatkov stikalo razbere, kdo je naslovnik in kdo pošiljatelj okvirja.

Ko stikalo iz nekega omrežnega segmenta prejme okvir, si zapomni naslov pošiljatelja. Ve namreč, da je naprava s tem naslovom dosegljiva preko tega omrežnega segmenta. V prihodnje bo promet za tega naslovnika pošiljalo le v naučeni segment, drugam pa ne.

Če stikalo ciljnega naslova ne najde med naučenimi, potem okvir razpošlje v vse segmente (angl. *broadcast*)[11]. Na ta način okvirji dosežejo tudi naslovnike, katerih naslova stikalo ne pozna.

Na sliki 6 so stikala B1, B2 in B3.

7.2.4 Usmerjevalniki

Stikala rešujejo problem neposredne povezave večjega števila zaključnih točk, ne omogočajo pa učinkovitega dostopa do drugih točk – do tistih, ki niso neposredno povezane s stikalom. Če se stikalo namreč ni naučilo, v kateri mrežni segment spada naslovnik, potem pošlje podatkovni okvir na vse vmesnike. To je potratno in nevarno.

Težavo rešujejo usmerjevalniki. Zopet moramo omeniti referenčni model OSI. Usmerjevalniki v njem delujejo na tretjem nivoju ali L3. Ta nivo se imenuje tudi omrežni nivo. Naloga usmerjevalnika je, da usmeri pakete (z novim nivojem je novo tudi izrazoslovje) proti naslovniku. Če usmerjevalnik ne ve kako do naslovnika, potem za razliko od stikala, paketa ne pošlje na vse strani, ampak proti vnaprej določenemu privzetemu prehodu (angl. *default gateway*).

Danes najpogostejši protokol tretjega nivoja je IP (angl. *Internet Protocol*) oziroma bolj natančno protokol IPv4 (angl. *IP version 4*). Kljub velikim naporom nekaterih se njegov naslednik, protokol IPv6, le počasi uveljavlja. Za nas razlike med njima niso bistvene. Grobo gledano protokol IPv6 razširja naslovni prostor protokola IPv4 z 2^{32} na 2^{128} bitov.

Usmerjevalnik pri posredovanju prometa proti cilju uporablja ciljni naslov (angl. *destination*

11 Pogosto se uporablja tudi izraz „poplavljanje“ (angl. *flooding*).

IP address, DIP) in usmerjevalne tabele (angl. *routing tables*). Te se lahko zgradijo samodejno[12], kot smo to videli pri stikalih, lahko pa so ročno nastavljene.

V primeru težav z dostavo paketa usmerjevalnik pošlje obvestilo o težavi pošiljatelju. Naslov pošiljatelja se torej ne uporablja za učenje, ampak za posredovanje sporočil.

Na sliki 6 imamo dva usmerjevalnika – R1 in R2.

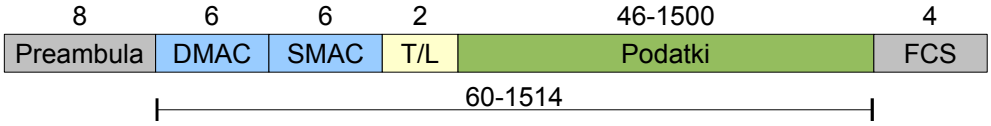
7.3 Omrežni protokoli

Vsak protokol je dogovor, ki ga spoštujejo vse stranke v pogovoru in tako omogoča sodelovanje med njimi. Omrežni protokoli omogočajo sodelovanje različnih naprav v omrežju. Nekaj omrežnih protokolov smo doslej že srečali.

7.3.1 Protokol Ethernet

Protokol Ethernet je nastal leta 1973 [1], od takrat pa njegova uporaba stalno raste. V manjših omrežjih ima popolno prevlado nad drugimi protokoli, vedno bolj uveljavljen pa je tudi v večjih omrežjih.

Protokol Ethernet omogoča pošiljanje okvirjev različnih velikosti, vendar ne manj kot 60 zlogov od začetka ciljnega MAC naslova pa do konca podatkovnega dela. Obstaja tudi zgornja meja, a ta ni povsem enoumno določena. Običajno je 1514 zlogov, včasih tudi več, celo do 9000 zlogov (angl. *jumbo frame*).



Slika 8: Celotna zgradba okvirja Ethernet.

Slika 8 prikazuje celotno zgradbo okvirja Ethernet, vendar se običajno ukvarjamo le s srednjim delom. Preambulo in blokovni kontrolni niz (angl. *Frame Check Sequence*, FCS) običajno proizvaja in preverja strojna oprema. Okvirjev z okvaro strojna oprema ne preda v obdelavo. Zanje izvemo le preko ustreznih števcev, ki jih ponuja strojna oprema.

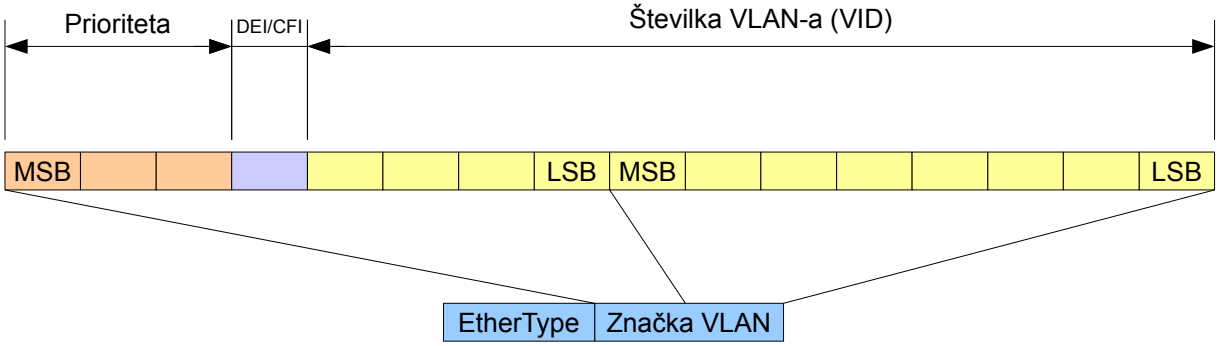
V nadaljevanju se bomo zato ukvarjali le z okvirjem Ethernet, kot smo ga predstavili na sliki 7, torej poenostavljeno različico brez preambule in blokovnega kontrolnega niza.

7.3.1.1 Naslov MAC

Zgradbo okvirja Ethernet smo predstavili že na sliki 7, zdaj predstavljamo še naslov MAC[13]. Sestavlja ga 6 zlogov, najmanj pomemben bit prvega zloga pa označuje, da naslov pripada v

12 Za samodejno gradnjo usmerjevalnih tabel se uporabljajo protokoli BGP (angl. *Border Gateway Protocol*), RIP (angl. *Routing Information Protocol*), OSPF (angl. *Open Shortest Path First*) in DHCP (angl. *Dynamic Host Configuration Protocol*).

13 Bolj splošno poimenovanje je naslov LAN [4] (angl. *LAN address*), vendar se v primeru omrežja Ethernet ta uporablja le redko.



Slika 9: Zgradba naslova MAC.

skupino naslovov za razpršeno ali skupinsko oddajanje (angl. *broadcast* in *multicast*). Zgradba naslova MAC je prikazana na sliki 9.

Načeloma so naslovi MAC enoznačni. To so proizvajalci omrežne opreme dosegli tako, da je vsakemu od njih določen nabor prvih treh zlogov, znotraj tega pa vsak prosto razpolaga z naslovi. Podjetje Iskratel ima na primer registrirano kodo[14] `00:D0:50`. To pomeni, da za vse naslove oblike `00:D0:50:xx:xx:xx` vemo, da pripadajo omrežni opremi podjetja Iskratel. Nekatere organizacije imajo veliko večji nabor kod .

Za skupinsko oddajanje paketov protokola IPv4 je rezervirana koda `01:00:5E`. To pomeni, da je mogoče v okviru naslovov MAC pošiljanje na 2^{24} = 16.777.216 skupinskih naslovov, vsi pa so oblike `01:00:5E:xx:xx:xx`.

7.3.1.2 Tip/dolžina

Poleg izvornega in ciljnega naslova MAC okvir Ethernet vsebuje še posebno polje, ki lahko označuje tip vsebine ali pa dolžino uporabnega dela podatkovnega okvirja[15]. Od tod tudi njegovo ime: T/L (angl. *type/length*).

Kako poteka ločevanje med obema pomenoma? Ker je največja dolžina koristne vsebine okvirja Ethernet 1500 zlogov velja, da vrednosti T/L polja do `0x600` pomenijo dolžino koristne vsebine, večje vrednosti pa tip okvirja[16].

7.3.1.3 Podatkovni del

Okvirji Ethernet brez preambule in blokovnega kontrolnega niza imajo minimalno velikost 60 zlogov. Ko izvzamemo še glavo okvirja Ethernet (6 zlogov ciljnega naslova, 6 zlogov izvornega naslova, 2 zloga polja T/L) pridemo do najmanjše velikosti podatkovnega dela 46 zlogov.

Če imamo torej en sam zlog, ki ga želimo prenesti uporabljajoč protokol Ethernet, ne gre drugače, kot da dejansko pošljemo vsaj 46 zlogov. Prejemnik je tisti, ki bo moral nekako

14 V jeziku organizacije IEEE se ta koda imenuje OUI (angl. *Organizationally Unique Identifier*). Iskanje po kodah je mogoče na internetu na strani `http://standards.ieee.org/regauth/oui/`.

15 Protokol Ethernet to polje v resnici vedno uporablja kot oznako za tip vsebovanega prometa. Kasnejša protokola IEEE 802.2 in IEEE 802.3 pa sta pomen tega polja razširila.

16 Trenutni spisek je dostopen na internetu na strani `http://www.iana.org/assignments/ethernet-numbers`.

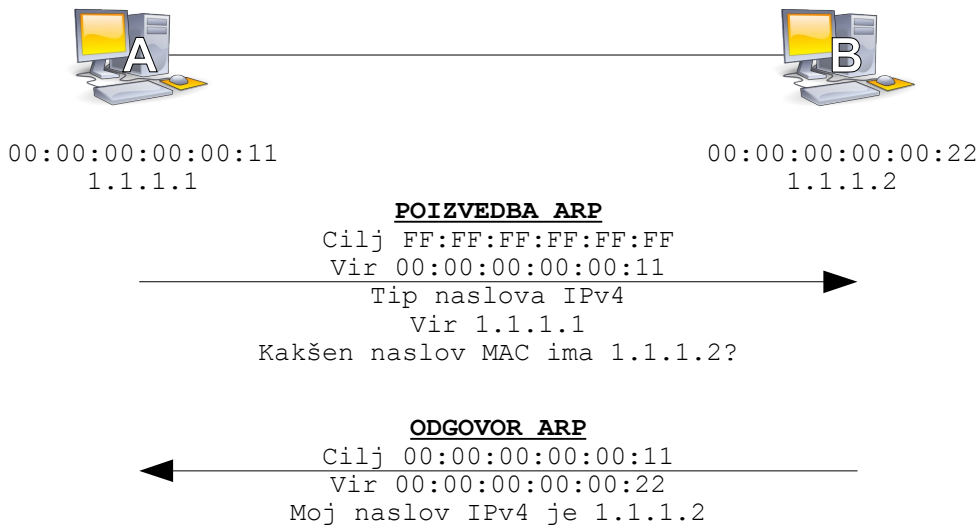
razumeti, da je pošiljatelj poslal le en zlog. Večina protokolov na višjih nivojih težavo rešuje tako, da imajo glavo znane velikosti, v njej pa zapisano velikost podatkovnega dela. Na ta način prejemnika lahko pravilno razume sprejeti okvir.

Glede podatkovnega dela ne gre prezreti opozorila, ki je namenjeno predvsem razvijalcem raznih protokolov. Nekateri protokoli uporabljajo glavo znane velikost in v njej zapišejo velikost celote, drugi uporabljajo kakšen drug pristop[17]. Zaradi napak, ki se lahko pojavijo v okvirju med prenosom ali pa zaradi namerno narobe oblikovanega okvirja se lahko zgodi, da nepreviden uporabnik med obdelavo okvirja zaide čez njegov konec. Temu pojavu rečemo preplavljanje medpomnilnika (angl. *buffer overrun*).

Denimo, da smo prejeli okvir, velik 60 zlogov, za katerega verjamemo, da vsebuje paket IPv4. Ob razčlenjevanju ugotovimo, da je celotna glava paketa IPv4 velika 60 zlogov (torej toliko kot prejeti okvir). Ker je podatkovni del okvirja velik le 46 zlogov, glava paketa IPv4 pa bi naj bila velika 60 zlogov, je očitno nekaj narobe. Če ob razčlenjevanju tega ne upoštevamo, potem je obnašanje nepredvidljivo.

7.3.2 Protokol za prevedbo naslovov

Protokol za prevedbo naslovov (angl. *Address Resolution Protocol*, ARP, [11]) je eden od protokolov, ki jih potrebujemo za izvedbo drugih, višjih protokolov na omrežju Ethernet. Naloga protokola je prevajanje med naslovi MAC in naslovi višjih protokolov. Njegovo delovanje bomo spoznali na primeru, pomagali pa si bomo z omrežjem s slike 10.



Slika 10: Delovanje protokola za prevedbo naslovov.

Naš cilj je, da na računalniku A (`1.1.1.1`) preverimo povezljivost z računalnikom B (`1.1.1.2`). Za to bomo uporabili program `ping`:

```
user@A:~/$ ping 1.1.1.2
```

Da bi preveril povezljivost, bo A poslal posebno vrsto paketa IP računalniku B, ta pa bo

17 Zelo pogosta je uporaba zapisa TLV (angl. *Type-Length-Value*). Pošiljatelj denimo v prvem zlogu zapiše, kakšen je tip podatka, v drugem, koliko je podatkov, sledijo pa podatkovni zlogi.

odgovoril nazaj računalniku A.

Ker sta povezana z omrežjem Ethernet (tudi neposredna povezava je vrsta omrežja), morata oba izvedeti za naslove MAC drug drugega. Paketi IP se namreč vedno pošiljajo točno določenemu naslovniku[18], v primeru omrežja Ethernet pa je za naslavljanje uporabljen naslov MAC.

Ker bomo program `ping` pognali na računalniku A, bo ta tudi začel s poizvedovanjem. V omrežje bo poslal posebno vrsto paketa ARP (zgradba paketa ARP je prikazana na sliki 11). Ker ne ve, komu poslati paket, ga bo poslal vsem na mrežnem segmentu v upanju, da bo nekdo odgovoril.

Ko bo ta paket sprejel računalnik B, bo prepoznal, da je poizvedba namenjena njemu. Zapomnil si bo, da je od računalnika z naslovom IPv4 `1.1.1.1` prejel okvir Ethernet, katerega izvorni naslov MAC je `00:00:00:00:00:11`. Zdaj B ve, kako preko omrežja Ethernet doseči naslov `1.1.1.1`.

Slika 11: Paket ARP.

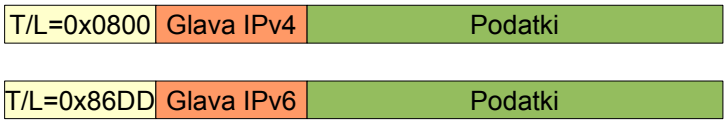
Sledi priprava odgovora, s katerim bo B obvestil računalnik A, da je je on tisti, ki sliši na naslov IPv4 `1.1.1.2`. Odgovor bo prišel v okvirju Ethernet, poslanem neposredno na naslov MAC `00:00:00:00:00:11`, iz katerega bo A razbral, da je pošiljateljev naslov MAC `00:00:00:00:00:22`. Zdaj tudi A ve, kako doseči naslov `1.1.1.2`.

S pomočjo pravkar opisanega postopka vsak računalnik zgradi tabelo za prevedbo naslovov (angl. *ARP Table*). Vnosi v tej tabeli sčasoma zastarajo. Takrat se postopek ponovi.

7.3.3 Protokola IPv4 in IPv6

Že pri predstavitvi protokola za prevedbo naslovov smo omenili internetni protokol (angl. *Internet Protocol*, IP). Obstajata dve različici tega protokola:

- Različica 4 (angl. *IP version 4*, IPv4) je starejša in bolj razširjena ([12]).
- Različica 6 (angl. *IP version 6*, IPv6) je novejša in še v postopnem uvajanju ([13]).



Slika 12: Paketa IPv4 in IPv6 v okvirju Ethernet.

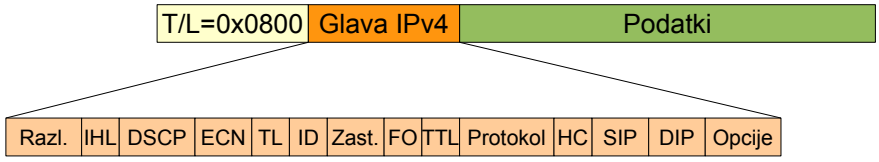
Med njima je kar nekaj razlik, ki pa za nas ta hip niso pomembne. Kadar uporabljamo izraz „IP“, imamo s tem v mislih IPv4 in IPv6. Vsi postopki, ki jih bomo prikazali na primeru protokola IPv4, veljajo tudi za protokol IPv6, le da so tam naslovi namesto 32-bitni 128-bitni.

18 Izjema sta razpršeno pošiljanje, pri katerem je cilj poseben naslov MAC za razpršeno pošiljanje (`ff:ff:ff:ff:ff:ff`) in skupinsko pošiljanje, pri katerem je cilj poseben naslov MAC, ki ga izračunamo na podlagi naslova IP-skupine.

Bistvo obeh protokolov je enako: omogočata usmerjanje prometa od izvora do ponora na podlagi internetnega naslova. Usmerjanje se izvaja na podlagi usmerjevalnih tabel.

Slika 12 prikazuje paketa IPv4 in IPv6 znotraj okvirja Ethernet. Polja paketa IP so predstavljena na sliki 13:

- Version – Različica protokola IP. V primeru IPv4 je vrednost `4`.
- IHL – velikost glave protokola IPv4 (angl. *Internet Header Length*) v korakih po 4 zloge. Najmanjša vrednost tega polja je 5 (najmanjša glava IPv4 obsega 20 zlogov), največja pa 15 (torej je največja glava IPv4 lahko velika 60 zlogov).
- DSCP – polje za ločevanje storitev (angl. *Differentiated Services Code Point*).
- ECN – obvestilo o nasičenju (angl. *Explicit Congestion Notification*).
- TL – dolžina celotnega paketa, vključno z glavo IPv4 (angl. *Total Length*).
- ID – identifikacija. To polje se uporablja ob razstavljanju in ponovnem sestavljanju datagramov.
- Flags – zastavica, ki se uporablja pri razstavljanju in ponovnem sestavljanju datagramov.
- FO – odmik kosa (angl. *Fragment Offset*) se uporablja pri razstavljanju in ponovnem sestavljanju datagramov.
- TTL – življenjska doba (angl. *Time To Live*) predstavlja število „skokov“ (vsak prehod usmerjevalnika šteje za en skok), ki jih bo paket preživel. Vsak usmerjevalnik število zmanjša za 1. Ko doseže vrednost 0, se paket zavrže.
- Protocol – vsebovani protokol, tip podatkovnega dela paketa IP.
- HC – kontrolna vsota glave IPv4 (angl. *Header Checksum*) se uporablja za odkrivanje napak v glavi.
- SIP – izvorni naslov IP- (angl. *source IP address*) paketa. To je naslov IP-zaključne točke, ki je paket odposlala.
- DIP – ciljni naslov IP- (angl. *destination IP address*) paketa. To je naslov IP-zaključne točke, ki bo paket prejela.
- Options – dodatni podatki. To polje je lahko tudi odsotno (to pomeni, da ima polje IHL vrednost `5`).



Slika 13: Zgradba paketa protokola IPv4.

7.3.3.1 Življenjska doba paketa protokola IP

Ko smo si ogledali okvir Ethernet (slika 8), nismo nikjer zasledili ničesar, kar bi mu omejevalo življenjsko dobo. Zanke v omrežju Ethernet so bile zato na začetku prepovedane,

danes pa ni več tako. Zanka namreč poveča robustnost omrežja (če je prekinjena ena veja, lahko promet še vedno teče po drugi). Da bi se izognili problemu okvirjev, ki brez konca krožijo po takih zankah, so razvili posebne protokole (družina protokolov STP [17], EAPS).

Če tega nimamo, potem paket, ki zaide v zanko, v njej ostane za vedno (most A pošlje paket mostu B, ta mostu C, ta zopet A, ta spet naprej B itd.). Počasi to pripelje do točke, ko preko omrežja ne moremo več prenašati koristnega prometa.

Bistvo omrežja IP je robustnost, zato so bile večkratne poti med točkami vključene že v zasnovo protokola. Da pa paket vseeno ne bi brez konca taval po omrežju (to je lahko posledica napačne nastavitve že enega samega usmerjevalnika), vključujejo paketi IP posebno polje, v katerem je zapisana življenjska doba paketa (angl. *Time To Live*, TTL).

Vsak usmerjevalnik, preko katerega gre paket IP, zmanjša vrednost v polju TTL za 1. Ko ta pade na 0, se paket zavrže, pošiljatelju pa se pošlje paket ICMP z obvestilom o razlogu, zaradi katerega je bil paket zavržen.

Prav to izrablja orodje `traceroute`. Pošilja pakete z zahtevo za odgovor proti ciljni zaključni točki, ob tem pa vrednost polja TTL postopoma povečuje od 1 do 255. To povzroči, da po vsakem koraku naslednji usmerjevalnik odgovori s paketom ICMP, `traceroute` pa na podlagi tega zgradi graf poti.

7.3.3.2 Vsebovani protokol

Protokol IP ni sam sebi namen. Rešuje problem usmerjanja prometa po spreminjajočem se omrežju. Protokoli višjih nivojev to uporabljajo tako, da svoje podatke vključijo v podatkovni del paketa IP. Ko sprejemnik obdeluje prejeti paket, se na podlagi identifikatorja vsebovanega protokola odloči, kaj storiti naprej. Identifikator je velik en zlog in tako dopušča do 256 različnih vsebovanih protokolov.

Najbolj znani protokoli, izvedeni na ta način, so ICMP, TCP, UDP in IGMP.

7.3.3.3 Naslov IP in sistem domenskih imen

Naslov IP je enoznačen[19] številčni naslov, ki je določen posamezni končni točki. V primeru IPv4 je to 32-bitni naslov, ki ga največkrat vidimo zapisanega v decimalni obliki: `1.2.3.4`. V primeru IPv6 je naslov 128-biten, za zapis pa se je uveljavila šestnajstiška oblika: `0123:4567:89ab:cdef:0123:4567:89ab:cdef`.

Kot vidimo, je naslov IPv4 še razmeroma preprost in omogoča pomnjenje, pri IPv6 pa je to zelo težko. Večinoma raje uporabljamo ljudem prijaznejša imena, recimo `www.iskratel.si`, `www.fri.uni-lj.si`, `slashdot.org` itd. Preslikavo med prijaznejšo in številčno obliko izvaja sistem domenskih imen (angl. *Domain Name System*, DNS).

7.3.3.4 Usmerjevalna tabela

Usmerjevalniki prejete pakete IP pošiljajo proti cilju s pomočjo usmerjevalne tabele (angl. *routing table*). Gre za urejen spisek pravil, ki imajo obliko `(razred, maska, vmesnik)`, uporabljajo pa se takole:

```
if (DIP & maska == razred) then pošlji_na(vmesnik)
```

19 Protokol za prevajanje omrežnih naslovov (angl. *Network Address Translation*, NAT) to trditev postavlja na laž. Tam je naslov enoznačen le znotraj podomrežja NAT. Do končnih točk v drugih podomrežjih NAT pridemo preko njihovih javno preslikanih naslovov.

Del usmerjevalne tabele je tudi privzeti prehod in sicer v obliki `(0.0.0.0, 0.0.0.0, privzeti_vmesnik)`.

Usmerjevalna tabela je urejena po naraščajoči splošnosti pravil. Najbolj natančna pravila so pri vrhu, in tako pridejo prej na vrsto, zadnje pa je pravilo za privzeti prehod. Pravilo A je bolj natančno od pravila B, če ima maska pravila A vključen vsaj en bit več kot maska pravila B:

```
A < B := (A.maska ≠ B.maska) & ((A.maska - B.maska) > 0)
```

Pri tem je treba opozoriti, da mora bitno gledano maska ustrezati regularnemu izrazu `1*0*`.

7.3.3.5 Usmerjanje prometa

Usmerjevalnik za prejeti paket najprej ugotovi, ali je namenjen njemu ali komu drugemu. Če je namenjen usmerjevalniku, potem ga preda lokalnemu mrežnemu skladu, ki ga ustrezno obdela. Če pa je namenjen drugam, potem s pomočjo usmerjevalne tabele ugotovi, kam, to je preko katerega vmesnika, naj ga odda.

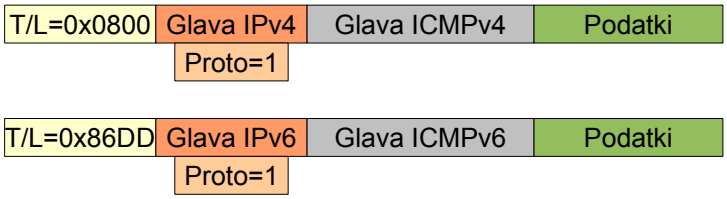
Usmerjevalnik po vrsti pregleduje pravila v usmerjevalni tabeli, to pa počne, dokler pri prvem ne pride do ujemanja. Običajno zadnje pravilo pripada privzetemu prehodu. To pravilo je zaradi svoje oblike vedno uspešno:

```
if (DIP & 0.0.0.0 == 0.0.0.0) then pošlji_na(privzeti_prehod)
```

Če po pregledu usmerjevalne tabele še vedno ne ve, kam poslati paket, potem usmerjevalnik pošiljatelja s paketom ICMP obvesti, da tega ni mogel opraviti.

7.3.4 Protokola ICMPv4 in ICMPv6

Podobno kot pri protokolih IP tudi pri protokolu ICMP obstajata dve različici. Vezani sta na protokol nižjega nivoja, v katerega sta ovita. Protokol ICMPv4 se uporablja v kombinaciji s protokolom IPv4, protokol ICMPv6 pa skupaj s protokolom IPv6. Po svoji funkcionalnosti sta si protokola zelo podobna, zato ju bomo obravnavali skupaj.



Slika 14: Paket ICMPv4 v paketu IPv4 in paket ICMPv6 v paketu IPv6.

Protokol ICMP je namenjen diagnostiki in odkrivanju napak v omrežju IP. Zgradba paketa ICMP je prikazana na sliki 14.

Najpogosteje uporabljane funkcije so:

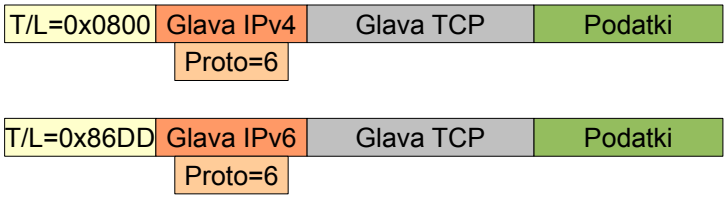
- *Echo request/reply* – zahteva za odgovor in odgovor. Ti dve funkciji predstavljata jedro programa `ping`. Prva zahteva od neke zaključne točke, naj odgovori, druga pa predstavlja odgovor.
- *Destination unreachable* – sporoča prejemniku, da paket, ki ga je poslal, iz različnih

razlogov ni dosegel cilja.

- *Traceroute* – zahteva za pošiljanje informacij.

7.3.5 Protokola TCP, UDP

Protokol za krmiljenje prenosa (angl. *Transmission Control Protocol*, TCP) je tisti, ki poskrbi, da se povezava med dvema točkama zdi kot povezava, čeprav se podatki v resnici prenašajo po paketih, ti pa lahko od izvora do cilja potujejo po različnih poteh, se izgubljajo in na cilj pridejo v spremenjenem vrstnem redu. Protokol TCP nad sicer paketnim omrežjem ustvari privid povezave.

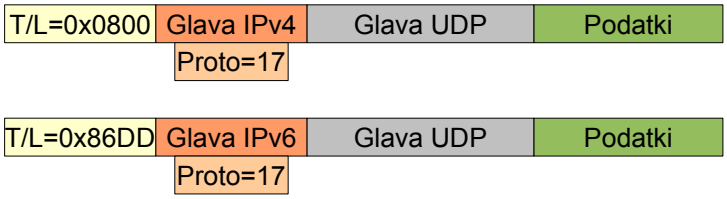


Slika 15: Paket TCP v paketu IPv4 in IPv6.

Slika 15 prikazuje paket TCP ovit v paket IPv4 ali IPv6, vse pa znotraj okvirja Ethernet.

Protokol uporabniškega datagrama (angl. *User Datagram Protocol*, UDP) je drugačen. Ne ustvarja vtisa povezave, vseeno pa omogoča prenašanje podatkovnih paketov do velikosti 64 KB. Po potrebi paket razstavi na manjše dele in jih na cilju znova sestavi. V primeru okvar ali izgub delov paketa se izgubi celoten paket, uporabnik pa o tem ne prejme nobenega obvestila. Zgradba paketa je prikazana na sliki 16.

Večina podatkov v današnjih omrežjih se prenaša preko protokolov TCP in UDP. Odpiranje spletne strani običajno povzroči odprtje več povezav TCP. Vsaka seja `telnet` ali `ssh` prav tako predstavlja po eno povezavo TCP. Prenos preko protokola FTP se dogaja v dveh povezavah TCP.



Slika 16: Datagram UDP v paketu IPv4 in IPv6.

Protokol UDP uporabljajo navidezna zasebna omrežja (angl. *Virtual Private Network*, VPN), strežniki DNS in še mnogi drugi.

Protokola TCP in UDP se med seboj precej razlikujeta, s stališča uporabe pa gre poudariti predvsem dve razliki:

- Protokol TCP ohranja vrstni red in celovitost poslanih in prejetih sporočil.
- Protokol UDP ne zahteva koordinacije med partnerjema v povezavi in je zato hitrejši.

Bolj globoko se v oba protokola ne bomo spuščali, saj za okvire tega dela to ni potrebno.

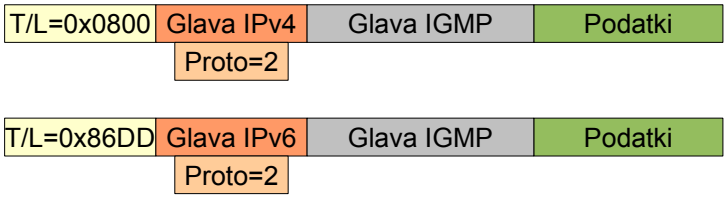
7.3.6 Protokol IGMP

Internetni protokol za upravljanje skupin (angl. *Internet Group Management Protocol*, IGMP, [10]) postaja pomemben šele v zadnjih letih. Gre za protokol, ki omogoča krmiljenje skupin za skupinsko pošiljanje na usmerjevalnikih in mostovih. Brez tega protokola bi promet, namenjen več naslovnikom, lahko dostavili samo na dva načina:

- promet bi pošiljali vsem naslovnikom; tisti, ki jim ni namenjen, bi ga odmetavali;
- promet bi že na izvoru pošiljali samo naslovnikom, ki so se predhodno prijavili.

Obe rešitvi imata pomanjkljivosti. Pošiljanje vsem naslovnikom pomeni v najboljšem primeru hudo potrato pasovne širine, v najslabšem pa varnostno luknjo, ko promet dobi tudi nekdo, ki mu ni bil namenjen. Recimo, da nekaj naročnikov plačuje dostop do televizije visoke ločljivosti (HDTV). Če bodo videotok prejemali tudi drugi, potem bodo neupravičeno gledali nekaj, za kar so ostali morali plačati. Bodo pa ob tem težko počeli karkoli drugega, saj bo videotok – ki ga mogoče niti ne želijo – zasedal večji del pasovne širine, ki so jo zakupili (1 Mb/s je za brskanje po spletu že kar sprejemljivo, a ne, če hkrati sprejemamo celoten video tok).

Če na izvoru pošiljamo promet samo zainteresiranim, smo s tem rešili prej omenjeni težavi, a si nakopali drugo. V času večernih poročil televizijo preko interneta v Sloveniji spremlja nekaj 100.000 ljudi. To bi pomenilo, da bi moral video strežnik pošiljati nekaj 100.000 videotokov. To seveda ni mogoče, saj je to za danes dostopno strojno opremo enostavno preveč. Rešitev bi bila v večjem številu videostrežnikov, poleg tega pa bi ti morali biti geografsko razpršeni, saj bi v nasprotnem primeru problem samo odmaknili od izvora.



Slika 17: Paket IGMP v paketu IPv4 in IPv6.

Protokol za upravljanje skupin to rešuje s prijavljanjem in odjavljanjem v skupine. Odjemalci, ki jih zanima določen podatkovni tok (ni nujno, da gre ravno za video ali radio), pošljejo prijavo v skupino. Vsi člani skupine prejemajo isti podatkovni tok. Taka prijava potuje od odjemalca do strežnika, ob tem pa se vsi mostovi in usmerjevalniki na poti seznanijo z željo odjemalca po včlanitvi v skupino. Ko prijava pride do strežnika, ta začne s pošiljanjem vsebine po poti, po kateri je prišla prijava. Če prijava že prej naleti na usmerjevalnik ali most, na katerem je podatkovni tok že na voljo, potem ni potrebe po čakanju, da prijava pride do strežnika, ampak lahko odjemalcu podatkovni tok priskrbimo takoj.

Ob odjavi se zgodba ponovi. Ko se zadnji odjemalec na nekem mrežnem segmentu odjavi iz skupine, takrat prvi naslednji usmerjevalnik, most ali video strežnik preneha s pošiljanjem toka v segment.

Usmerjevalniki in mostovi tako prejemajo en sam podatkovni tok, oddajajo pa ga v vseh segmentih, v katerih je vsaj en prijavljen odjemalec.

Zgradba paketa IGMP je prikazana na sliki 17.

7.4 Omrežni sklad operacijskega sistema Linux

Omrežni sklad operacijskega sistema Linux si zasluži posebno omembo iz več razlogov:

- Njegova izvorna koda je prosto dostopna, zato je odličen študijski pripomoček. Predstavlja odlično mesto, kjer si lahko teoretične koncepte ogledamo tudi v praksi. Je laboratorij, kjer lahko z razmeroma majhnim vložkom preverjamo nove ideje.
- Linux je v osnovi zelo odprt operacijski sistem. Ne le da je na voljo celotna izvorna koda, tudi zasnova posameznih delov je taka, da predvideva preprosto sestavljanje z drugimi komponentami.
- Linux spada v družino operacijskih sistemov Unix. Zanje je značilno, da ponujajo na sistemskem nivoju veliko število specializiranih orodij. Vsako opravlja majhno nalogo, a je pri tem dobro. Z njihovim povezovanjem hitro sestavimo sklope, ki opravljajo zahtevnejše naloge.
  Ta filozofija se odraža tudi v jedru Linuxa, kjer so posamezne komponente specializirane za točno določeno nalogo, omogočajo pa enostavno sestavljanje v večje celote.
- Korenine zdajšnje izvedbe omrežnega sklada segajo v operacijski sistem BSD ([46]). Bolj ali manj vsi sodobni operacijski sistemi so dele ali celoto omrežnega sklada podedovali od njega.
- Je odličen demonstrator pristopov pri zgradbi modularnega sistema.

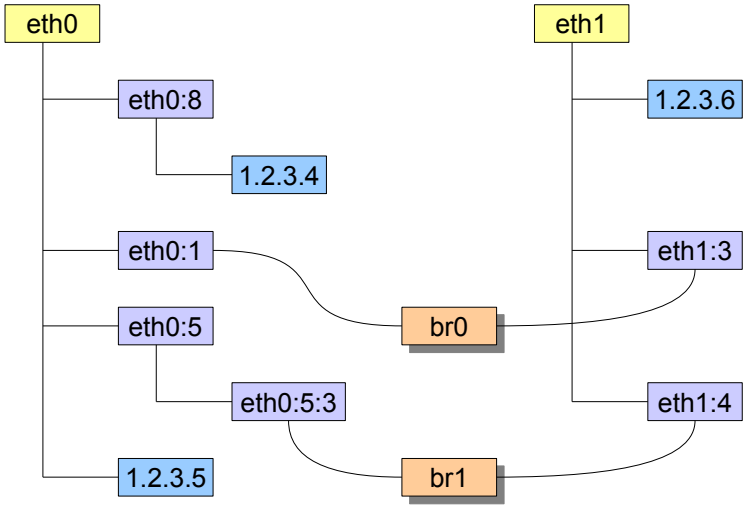
7.4.1 Most operacijskega sistema Linux

Most operacijskega sistema Linux je v primerjavi z drugimi tukaj omenjenimi najbolj preprost. Od vsega opisanega podpira le tabelo MAC naslovov in protokol STP za preprečevanje zank (menjava za kak drug protokol je zelo preprosta). Ne zaveda se označb VLAN, vendar pa ob pomoči ločenega modula za delo z značkami VLAN lahko dosežemo enak učinek kot pri mostu, ki se znač VLAN zaveda. Prav tako ne podpira filtrov, ima pa nastavke za uporabo različnih filtrov že jedro operacijskega sistema Linux.

Zaradi tako omejene funkcionalnosti si skoraj ne zasluži omembe v družbi ostalih mostov. Duh njegove odprtosti je razlog, da ga vseeno opisujemo. Ta je pomembno vodilo pri načrtovanju našega mosta.

7.4.2 Sestavljanje kock

Omrežni sklad Linux ni monolitna entiteta, ampak je bolj podoben kockam Lego, ki jih sestavljamo po potrebi.



Slika 18: Univerzalnost omrežnega sklada operacijskega sistema Linux.

Najlažje univerzalnost Linuxovega pristopa prikažemo na primeru. Slika 18 prikazuje kombinacijo štirih elementov omrežnega sklada Linux:

- `eth0` in `eth1` predstavljata omrežni napravi, dva vmesnika Ethernet.
- `eth0:8` je navidezna omrežna naprava. Navezana je na omrežno napravo `eth0`, od nje pa prejema okvirje, ki spadajo v VLAN 8. Če na `eth0:8` pošljemo neoznačen okvir, bo ta proti `eth0` oddal okvir, ki bo vseboval oznako VLAN s številko 8.
  Podobno `eth0:1` skrbi za VLAN 1 na vmesniku `eth0` in `eth0:5` za VLAN 5. Na vmesnik `eth1` sta navezani navidezni napravi `eth1:3` in `eth1:4`, ki tam prestrezata okvirje VLAN za VLAN ID 3 in 4.
- `eth0:5:3` je samo nadgradnja prej spoznanih navideznih vmesnikov VLAN. Gre za vmesnik, ki je navezan na `eth0:5`, prevzema pa okvirje, ki imajo znotraj VLAN glave s številko 5, še eno glavo VLAN, tokrat s številko 3. Ta navidezni vmesnik torej skrbi za dvojno označene okvirje.
- `1.2.3.5` je vmesnik IPv4, ki deluje neposredno na vmesniku `eth0`. To pomeni, da sprejema pakete IPv4, ki pridejo na vmesnik `eth0`, morajo pa biti brez oznak VLAN.
  `1.2.3.6` je enake vrste vmesnik, le da deluje na vmesniku `eth1`.
  `1.2.3.4` je tudi vmesnik IPv4, vendar deluje na navideznem vmesniku `eth0:8`. To pomeni, da pričakuje pakete IPv4, ki prihajajo na navidezni vmesnik `eth0:8` – torej so označeni z značko VLAN s številko 8.
- `br0` je most operacijskega sistema Linux, navezan na navidezna vmesnika `eth0:1` in `eth1:3`. To pomeni, da premošča promet, ki prihaja na vmesnik `eth0` in je označen z značko VLAN in številko 1, na vmesnik `eth1`, označena z značko VLAN in številko 3.
  `br1` je most operacijskega sistema Linux, navezan na navidezni vmesnika `eth0:5:3` in `eth1:4`. To pomeni, da premošča promet, ki prihaja z dvema značkama VLAN na vmesnik `eth0`, v enojno označenega na vmesniku `eth1`.

Za lažje razumevanje slike 18 še nekaj pripomb:

- Za naprave, označene z isto barvo, skrbi isti gonilnik[20]. Vsega skupaj imamo tako praviti s štirimi gonilniki:
  - `eth0` in `eth1` podpira gonilnik za strojno opremo.
  - `eth0:8`, `eth0:1`, `eth0:5`, `eth0:5:3`, `eth1:3` in `eth1:4` so navidezne naprave. Zanje skrbi gonilnik VLAN.
  - `br0` in `br1` sta navidezni napravi. Zanju skrbi gonilnik Linuxovega mosta.
  - `1.2.3.4`, `1.2.3.5` in `1.2.3.6` so vmesniki IPv4, za katere skrbi gonilnik IPv4.
- Po vseh povezavah na sliki teče promet v obe smeri. Vmesnik IPv4 `1.2.3.4` lahko sprejema in oddaja okvirje.

7.4.3 Princip čebule

Sestavljanje kock, kot je bilo delovanje Linuxovega omrežnega sklada prej opisano, je mogoče zato, ker lahko omrežne protokole predstavimo s prispodobo čebule. Vsak naslednji protokol predstavlja novo plast. Ko aplikacija pošlje paket proti oddaljenemu računalniku, se ta „spusti“ po omrežnem skladu, ob tem pa pridobiva vedno nove plasti – vsak protokol doda svojo. Ko se paket na drugi strani „dviga“ po omrežnem skladu, vsak protokol odstrani plast, ki jo je dodal, na koncu pa ciljna aplikacija dobi podatkovni paket, kakršnega je odposlala izvorna aplikacija. Omenjeni princip je prikazan tudi na sliki 4.

Tak način obravnave ni posebnost Linuxa, ampak gre za običajen pristop k obravnavi mrežnih protokolov.

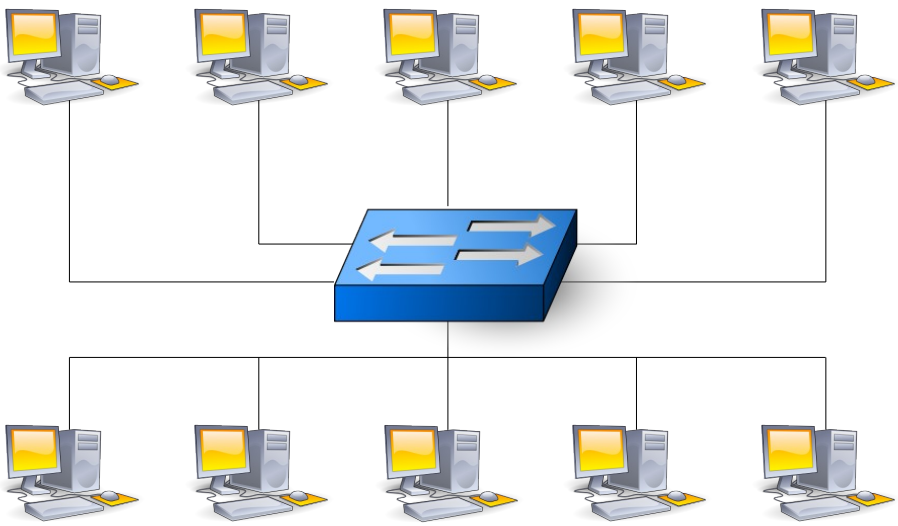
20 Predpostavljamo, da sta vmesnika `eth0` in `eth1` strojno enake vrste, sicer bi vsakega od njiju podpiral drug gonilnik.

8 Stikala in mostovi

V prejšnjem poglavju smo spoznali za nas najpomembnejšo strojno in programsko opremo, ki sestavlja računalniška omrežja. V tem poglavju se bomo posvetili stikalom in mostovom z namenom, da bolj podrobno spoznamo naloge, ki jih opravljajo, mimogrede pa bomo tudi spoznali, kaj vse stikala in mostovi za opravljanje svojih nalog potrebujejo.

8.1 Stikalo ali most

Prvotni izraz za stikalo ali most je danes komaj še znan. V podjetju Digital Equipment Corporation (DEC) so prvotno uporabili izraz *Data Link layer relay* [35].



Slika 19: Položaj stikala v omrežju.

Stikalo (angl. *switch*) in most (angl. *bridge*) sta izraza, ki se pogosto uporabljata izmenjaje. Enotne definicije enega in drugega ni, različni avtorji pa naštevajo več razlik.

- stikalo združuje točkovne povezave (zgornja polovica na sliki 19), most pa segmente omrežja (spodnja polovica na sliki 19, [3]);
- stikalo je izvedeno v strojni opremi, most pa programsko[21] [37];
- stikalo združuje enakovredne vmesnike, most pa uporabniške vmesnike priklaplja na hrbtenico [40];
- stikalo se omejuje na preklapljanje okvirjev med segmenti omrežja. Most temu dodaja še funkcije filtriranja, klasificiranja, oblikovanja prometa itd. [38].

21 V resnici je danes težko najti čisto rešitev. Programske uporabljajo različne nivoje strojnega pospeševanja, pri strojnih pa gre prav tako za kombinacijo prave strojne rešitve in mikrokodiranja.

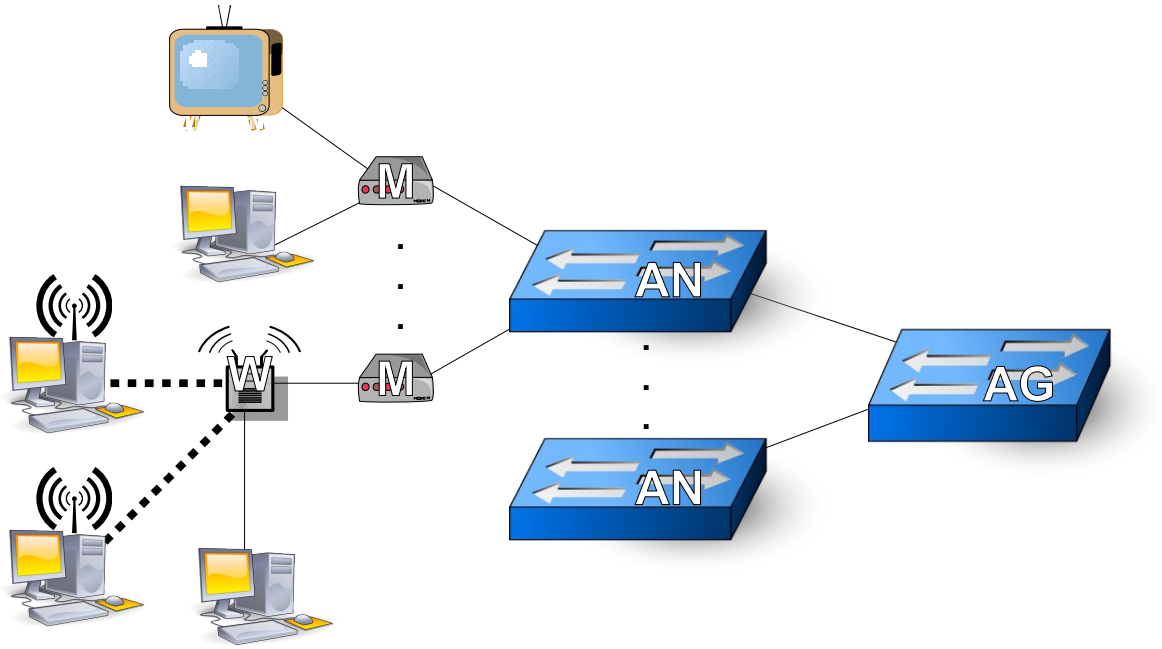
Najbolj enostavno je, če gledamo na stikala kot na strojno izvedene mostove. Vse ostale razlike imajo korenine v tem dejstvu: so hitrejši, a imajo bolj omejen nabor funkcionalnosti.

Omembe vredna posebnost je IEEE, ki v družini standardov 802.1 ([5],[6],[7],[8]) skoraj izključno uporablja izraz most, le izjemoma pa tudi stikalo. Pri tem v svojih definicijah ne predpisuje načina izvedbe (strojna, programska ali hibridna različica).

Če pri poimenovanju stikala in mostovi predstavljajo nekaj težav, pa je njihova vloga jasna: povezovanje na nivoju povezave (angl. *link layer*).

8.2 Vrste mostov

Mostovi so v naših omrežjih pogostejši, kot si to predstavljamo, nekaj jih je prikazanih na sliki 20:



Slika 20: Vseprisotnost mostov.

- Domači usmerjevalniki (W) pogosto vsebujejo most.
  Ti mostovi redko zmorejo več kot 100 Mb/s trajne prepustnosti (angl. *sustained rate*). Podpirajo širok nabor funkcionalnosti: požarne zidove, omejevanje dostopa, kombinirajo brezžična in fiksna omrežja itd.
- Mostovi poleg usmerjevalnikov IP predstavljajo jedro zasebnih omrežij.
  Kadar ne gre za prej omenjene domače usmerjevalnike, so to lahko zelo zmogljive naprave, ki vsebujejo požarne zidove, podporo za navidezna zasebna omrežja, razne vrste tunelov ...
- Most v taki ali drugačni obliki je ponavadi prisoten v opremi, ki je nameščena pri naročnikih (angl. *Customer-Premisses Equipment*, CPE). To so modemi DSL in

optični modemi (M).
Prepustnost teh mostov je prilagojena največji hitrosti povezave proti DSLAM-u. V primeru optičnih modemov je to 100 Mb/s, sicer pa manj. Ne podpirajo veliko funkcionalnosti. Pri njihovem načrtovanju in izdelavi je glavna cena, posledično pa vsebujejo „šibke“ procesorje.

- Naročniška oprema je povezana z dostopovnim vozliščem (angl. *Access Node*, AN). Ta ima od vseh naštetih mostov najširšo funkcionalnost[22].
  Ti mostovi morajo zagotavljati visoko prepustnost (zaželena je polna hitrost naročniških linij), obenem pa ponujati funkcije, ki so zahtevane v večjih omrežjih. To vključuje ščitenje pred raznimi nevarnostmi, delo z označitvami VLAN, opremljanje nekaterih protokolov z dodatnimi informacijami (PPP, DHCP), optimiziranje prometa, posredovanega več naslovnikom ... Spisek funkcionalnosti se stalno daljša.
- Sledi več nivojev mostov, ki promet zgostijo (AG).
  DSLAM deluje v hitrostnem razredu 1 Gb/s, na naslednjih nivojih pa mostovi morajo podpirati trajno prepustnost 10-100 Gb/s. To seveda gre na račun manj kompleksnih funkcij.

V okviru tega dela se bomo najbolj posvetili razredu mostov DSLAM. Ti so za nas najbolj zanimivi zaradi kombinacije razmeroma visoke prepustnosti in bogate funkcionalnosti.

8.2.1 Pasivna in aktivna omrežna oprema

Pogosto slišimo za delitev na pasivno in aktivno omrežno opremo. Najlažje to razložimo, če opredelimo kot aktivno omrežno opremo tisto, ki okvirje (pri višjih protokolih pa pakete) ne le posreduje, ampak jih lahko tudi spreminja [22]. Po teh merilih so prvi mostovi sodili med pasivno omrežno opremo, danes pa večinoma sodijo med aktivno.

Pojavlja se tudi zelo podoben izraz, *arhitektura aktivnega omrežja* (angl. *active network architecture*), ki pa gre še korak dlje. V skrajnem primeru gre za menjavo podatkovnih okvirjev, ki se pretakajo po omrežju, za *kapsule* – kose programov, ki se izvajajo na vozliščih, ki jih paket prehaja [23].

8.3 Delovanje mostov

Stikala in mostovi so elementi omrežja, namenjeni povezovanju segmentov in omrežnih naprav na OSI nivoju 2 (glej sliko 2).

8.3.1 Osnovne naloge

8.3.1.1 Preklapljanje prometa

Preklapljanje prometa med segmenti omrežja se izvaja na podlagi ciljnega naslova, v primeru omrežja Ethernet je to naslov MAC. Če je poizvedba po ciljnem naslovu uspešna – to pomeni, da vemo, na katerem vmesniku se naslovnik nahaja –, potem se okvir usmeri proti naslovniku.

22 Prej omenjeni mostovi so za širše funkcionalnosti strojno prešibki, poleg tega pa zaradi umestitve v uporabniško okolje ne uživajo zaupanja. Kasneje omenjeni mostovi morajo obdelovati bistveno večje količine prometa, zato so obseg funkcionalnosti krči na račun prepustnosti.

Če poizvedba ni uspešna (angl. *Destination Look-up Failure*, DLF), potem poskusimo naslovnika doseči s pošiljanjem okvirja v vse mrežne segmente[23] [16].

Če je cilj naslov za razpršeno pošiljanje, potem okvir prav tako pošljemo vsem vmesnikom[24]. Pri skupinskih ciljnih naslovih pa je obnašanje odvisno od funkcij, ki jih most podpira. Če obvlada skupinsko pošiljanje, potem bo okvir razposlan le vmesnikom, ki so prijavljeni v skupino, sicer pa vsem vmesnikom.

8.3.1.2 Gradnja usmerjevalnih tabel

Pogoj za usmerjanje na podlagi poznavanja naslovov MAC je gradnja usmerjevalnih tabel. Ta lahko poteka samodejno ali ročno, podatki pa so zbrani v tabeli naslovov MAC (angl. *MAC table*). V okviru družine standardov IEEE je bolj udomačen izraz *filtering database*.

Samodejna gradnja (učenje) poteka neprekinjeno. Ko na nekem vmesniku sprejmemo nov okvir, si zapomnimo njegov izvorni naslov in vmesnik, preko katerega smo paket sprejeli. Sklepamo, da bo naslovnik še nekaj časa dosegljiv v omrežnem segmentu, ki je dosegljiv preko tega vmesnika, zato bomo nanj naslovljene okvirje pošiljali nanj.

Pri samodejni gradnji usmerjevalnih tabel se ne učimo vseh naslovov. Na sliki 9 je prikazana zgradba naslova MAC. Če je v njem vključen bit, ki označuje razpršeno in skupinsko pošiljanje (to je najmanj pomemben bit prvega zloga), potem se naslova ne učimo samodejno.

Samodejno naučenim naslovom MAC rečemo tudi dinamični, saj je njihov vnos v usmerjevalno tabelo odvisen od dogajanja v omrežju. Most jih v usmerjevalno tabelo vnaša samodejno, prav tako pa jih bo tudi samodejno iz nje izbrisal.

Naslovom, ki smo jih vnesli ročno, rečemo statični – most jih ne bo pozabil. Operater, ki jih je vnesel, jih lahko seveda tudi izbriše.

Posebno kategorijo naslovov MAC predstavljajo skupinski naslovi. Njim se bomo posvetili kasneje.

Zaradi nadzora porabe pomnilnika lahko mostu omejimo število naslovov, ki se jih bo naučil. Omejitev je običajno globalna (za celoten most), lahko pa velja tudi za posamezni vmesnik. Slednje je še posebej prikladno, ker onemogoči enemu vmesniku, da bi izstradal vse ostale.

8.3.1.3 Staranje

Dinamični vnosi v usmerjevalni tabeli ne ostanejo za vedno, ampak se čez čas postarajo. Privzeto je to čez 5 minut, predvidene pa so vrednosti med 10 in 1.000.000 sekund [6]. Ko se posamični vnos postara, ga iz usmerjevalne tabele odstranimo.

Glavni razlog za staranje je omejena količina pomnilnika, ki ga imamo na voljo. Velika količina pomnilnika bi podražila most. S pomočjo pravilne izvedbe staranja poskrbimo, da imamo v tabeli naslovov MAC pogosto uporabljane naslove. Redko uporabljenih ne poznamo, zato bomo nanje naslovljeni promet pošiljali vsem vmesnikom.

Očitno sta hitrost staranja in velikost tabele naslovov MAC zelo pomembna pri doseganju dobrih prometnih lastnosti mosta. Če je staranje hitro in tabela majhna, potem bo večja količina prometa poslana vsem vmesnikom. V večini primerov to pomeni večje zakasnitve

23 Vse razen tistega, iz katerega smo okvir prejeli. Vračanje okvirja tja, od koder je prišel, bi bilo nesmiselno.

24 Na omrežnem nivoju 2 načeloma ni omejevanja razpršenega pošiljanja prometa, kot je to običajno na omrežnem nivoju 3. Razlog gre iskati v dejstvu, da je na tem osnovanih precej protokolov (BOOTP, DHCP, ARP ...). Zaradi varnosti pa lahko pri mostovih tudi to omejimo.

(pošiljanje vem vmesnikom pogosto pomeni večjo porabo časa v primeru s pošiljanjem preko enega samega vmesnika), hkrati pa tudi večjo porabo pasovne širine.

8.3.1.4 Prehod naprav med segmenti

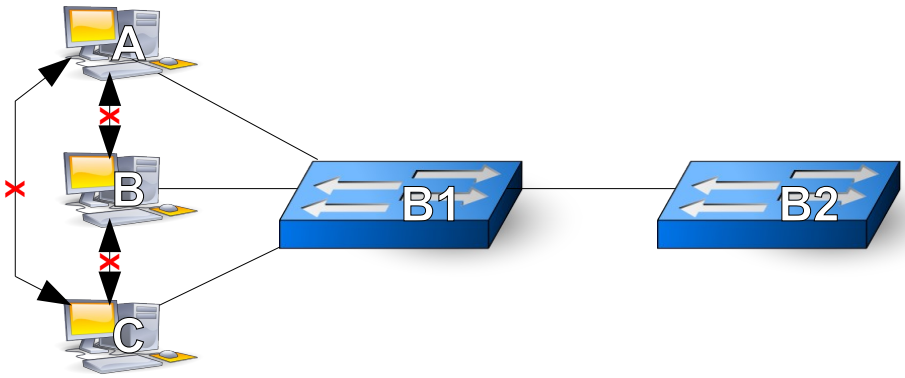
Naprave se s stališča mosta lahko premikajo. Razlog za to je lahko sprememba mrežne topologije (namesto po enem potem vidimo napravo preko drugega kraka omrežja), lahko pa tudi fizični premik naprave (nekdo se lahko s prenosnim računalnikom preklaplja na različne segmente omrežja). Tako obnašanje moramo predvideti in ustrezno ravnati, ko do njega pride.

Del problema rešuje že staranje. Ko bo vnos iz tabele naslovov MAC zaradi zastaranja odstranjen, bodo paketi za neznanega naslovnika do uspešnega učenja razposlani na vse vmesnike. Ko bo premaknjena naprava oddala prvi okvir, se bo most naučil, kje po novem najde napravo, in od takrat pošiljanje na vse vmesnike ne bo več potrebno.

Obstaja pa tudi možnost, da na vmesniku B sprejmemo okvir z izvornim naslovom, ki smo se ga naučili na vmesniku A. V tem primeru se lahko nemudoma naučimo naslov na vmesniku B, lahko pa tak dogodek obravnavamo kot napako in vmesnik B začasno blokiramo[25].

8.3.1.5 Skrivanje sosedov

Mostovi se uporabljajo v različnih tipih omrežij. V manjših okoljih mreže običajno sestavljajo enakovredne naprave, vloga mosta pa je, da jih na ta način tudi poveže. V večjih omrežjih, predvsem to velja za dostopna omrežja (angl. *access networks*), pa ne gre za povezovanje enakovrednih naprav. V resnici želimo le pripeljati promet od naročnikov (angl. *access ports*) do neke zaključne točke (običajno strežnik BRAS) in nazaj, ne želimo pa, da bi se naročniki neposredno videli med seboj[26] (slika 21).



Slika 21: Nevidni sosedi.

Most v takem primeru omogoči določanje vmesnikov, ki se med seboj vidijo – med njimi je promet mogoč –, in vmesnikov, ki se med seboj ne vidijo.

25 Tak dogodek je lahko posledica povsem normalnega početja (hiter prehod prenosnika iz enega v drugo omrežje), lahko pa gre za zlonameren poskus ponarejanja naslova MAC (angl. *MAC address spoofing*).

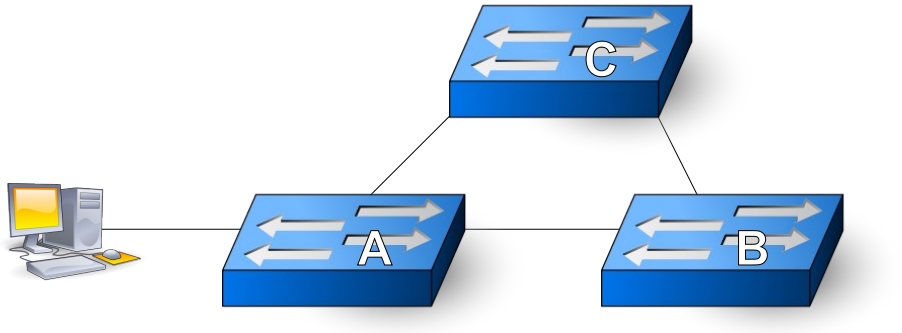
26 V resnici se uporabniki med seboj še vedno vidijo, a ne kot sosedje na mrežnem segmentu, pač pa preko dodeljenih naslovov IP. Dostopna omrežja so v večini primerov čista omrežja Ethernet, prehod v svet IP pa se zgodi šele na strežniku BRAS.

8.3.1.6 Prekinjanje zank

Na sliki 7 vidimo, da glava okvirju Ethernet vsebuje ciljni naslov MAC (DMAC), izvorni naslov MAC (SMAC) in tip oziroma dolžino (T/L). Nobenega od teh polj most ne spreminja.

Če v omrežju obstajajo obroči, tako kot na sliki 22, potem obstaja nevarnost, da bo že majhna količina prometa omrežje povsem zapolnila.

Če računalnik na sliki 22 pošlje en sam okvir proti mostu A, potem bo ta v najboljšem primeru poslal okvir prosti mostu B ali C, v najslabšem pa proti B in C. Kakorkoli že, mosta B in C bosta prejeti okvir poslala na preostale vmesnike in tako bo okvir kmalu zopet pri mostu A. Če ta nima vklopljene zaščite proti prehajanju med segmenti, bo okvir zopet poslal naprej. Na ta način okvir brez konca[27] kroži po omrežju, s tem pa zaseda pasovno širino. Ker so tovrstne topologije pogoste (glavni razlog je zanesljivost v primeru odpovedi stikala ali povezave), je nujno potreben način, da tako postavitev najprej odkrijemo, nanjo reagiramo, ob spremembi topologije pa to čim hitreje zaznamo.



Slika 22: Zanka v mrežni topologiji.

Najprej je ta vloga pripadla nizu protokolov, ki jih danes označujemo kot xSTP. Prvi je bil protokol STP, sledila sta RSTP, ki je prinesel hitrejše odzive na spremembe v topologiji omrežja in MSTP, ki vključuje podporo za označbe VLAN.

Kot hitrejši alternativi – hitrost preklopa je zelo pomembna – sta v uvajanju protokola EAPS in ERPS, nekateri proizvajalci pa ponujajo tudi svoje zasebne rešitve.

8.3.1.7 Upravljanje

Za konec puščamo še eno zelo pomembno nalogo, ki jo mora most tako ali drugače podpirati: upravljanje. Nekateri mostovi so glede tega zelo omejeni in ponujajo le malo nastavitev in malo podatkov o prometu, drugi mostovi imajo obojega veliko.

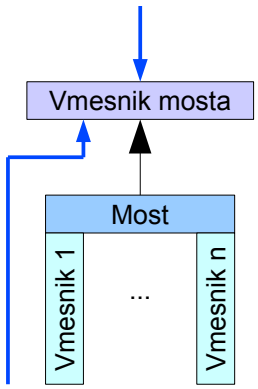
Glavno vprašanje v zvezi z upravljanjem je, kako je upravljalni vmesnik mosta dosegljiv. Obstajata dva pristopa:

- Upravljanje s souporabo vmesnikov. Upravljalni promet gre po isti poti kot ostali promet (angl. *in-band*). Na sliki 23 to predstavlja modra puščica, ki prihaja od spodaj

27 Za 100 BASE-TX je BER (angl. *Bit Error Rate*) 10^{-10}, torej 1 bit od 10^{10} [32]. Torej bi čez čas okvir le umrl, saj bi se vanj prikradla vsaj ena bitna napaka, to pa bi nivo MAC zaznal preko kontrolne vsote (FCS) in okvir zavrgel.
Usmerjevalniki IP te težave nimajo, saj glava IP vsebuje polje TTL, ki ga vsak usmerjevalnik zmanjša za 1. Ko pade na 0, usmerjevalnik paket zavrže.

in prečka most.

- Upravljanje preko ločenega vmesnika. Upravljanju mosta so namenjeni posebni vmesniki, prometni in upravljalni promet pa se ne križata (angl. *out-of-band*). Na sliki 23 to predstavlja modra puščica, ki prihaja od zgoraj in ne prečka mosta.



Slika 23: Upravljanje preko ločenega vmesnika ali s souporabo običajnih vmesnikov.

Prednost prvega načina je predvsem v lažji postavitvi omrežja – drugi pristop namreč zahteva gradnjo vzporednega omrežja, ki je namenjeno izključno upravljanju. Prednost drugega pristopa pa je povečana varnost, saj tudi v primeru napak pri nastavitvah omrežja ni možnosti, da bi se upravljalni promet pomešal z običajnim. Tako je nevarnost vdora in zlorabe omrežne opreme veliko manjša.

Pogosto most ponuja obe možnosti, z nastavitvami pa lahko izklopimo mešanje upravljalnega in običajnega prometa.

8.3.2 Označevanje VLAN

V manjših omrežjih opisano zadostuje, večja pa z uporabo označevanja VLAN [6] razširijo naslovni prostor in nabor funkcionalnosti.

Če je označevanje VLAN uporabljeno, potem navidezno razdeli omrežje na več podomrežij. V tem primeru mostovi povezujejo segmente podomrežij, ponujajo pa tudi možnost prehajanja okvirjev iz enega podomrežja v drugo.

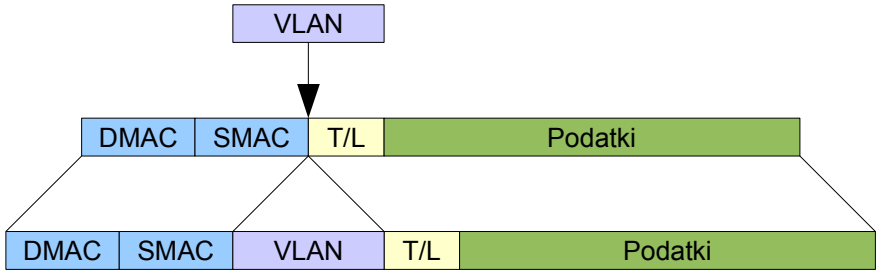
Pristopna omrežja (angl. *access networks*) predstavljajo mejo omrežja, ki ga upravlja in uporablja posamezen operater. To pomeni, da se nanj neposredno priklapljajo uporabniki – poslovni in zasebni. Operaterji zato že v prvi vrsti[28] postavljajo različne zaščitne mehanizme, ki omogočajo lažji nadzor nad omrežjem, omejujejo dostop do storitev, obenem pa odstranjujejo različne grožnje, ki omrežju grozijo od zunaj.

Uvedba označevanja VLAN vpliva ne vpliva neposredno na tabelo naslovov MAC. Če pa želimo bolj optimalno in varno delovanje, pa se mora učiti naslove po posameznih VLAN-ih, prav tako pa vse poizvedbe v morajo vsebovati VLAN ID. Tabela naslovov MAC upošteva le vnose, ki spadajo v VLAN, po katerem povprašuje most.

28 Prvo vrsto v resnici pogosto predstavljajo modemi, ki jih operaterji namestijo pri naročnikih. Ker pa modem fizično ni pod nadzorom operaterja, zanj velja, da mu ne moremo povsem zaupati. Uporabnik lahko vanj poseže ali ga zamenja.

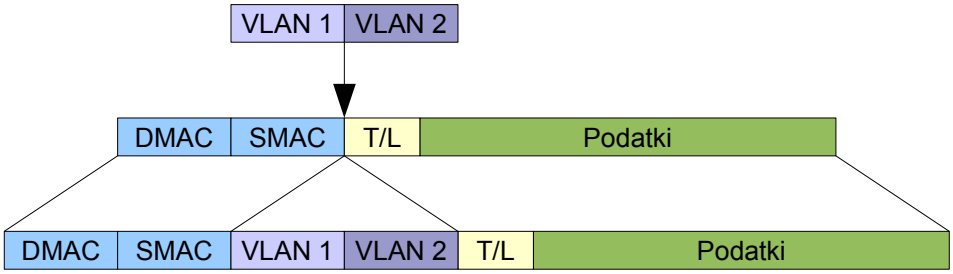
8.3.2.1 Oblika okvirjev VLAN

Označevanje VLAN izvajamo tako, da med glavo okvirja Ethernet in preostanek paketa vrinemo glavo VLAN. Shematski prikaz takega vrivanja je prikazan na sliki 24.



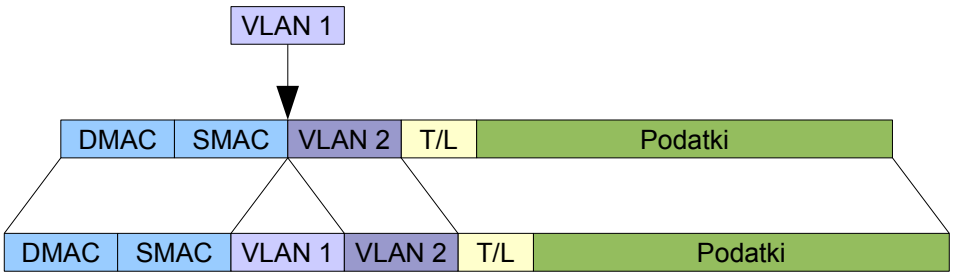
Slika 24: Zgradba okvirja Ethernet, ki vključuje eno glavo VLAN.

Načeloma lahko vrinemo poljubno število označitev VLAN, se pa moramo zavedati, da vsaka označitev za 4 zloge zmanjša prostor, ki je na voljo ostalim podatkom. V praksi se veliko uporablja enojno in dvojno označevanje.



Slika 25: Vstavljanje dveh glav VLAN v neoznačen okvir Ethernet.

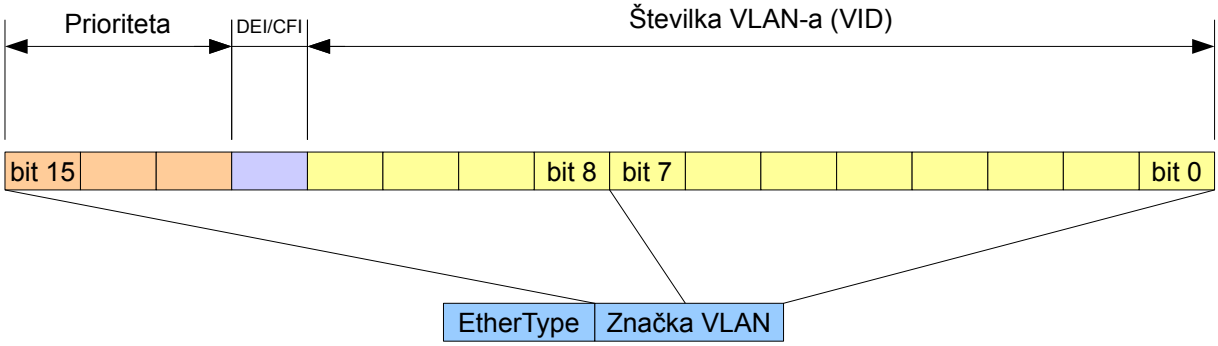
Pri enojnem označevanju okvir Ethernet vsebuje eno glavo VLAN (slika 24), pri dvojnem pa dve (sliki 25 in 26).



Slika 26: Vstavljanje ene glave VLAN v enojno označen okvir Ethernet.

8.3.2.2 Naslovni prostor

Zaenkrat ni strahu, da bi v kratkem začelo primanjkovati naslovov MAC. Operaterji večjih omrežij s tem naslavljajo potrebo po delitvi fizičnega omrežja na logična podomrežja glede na vrsto funkcionalnosti. V primerih, ko na istem fizičnem omrežju gosti več ponudnikov



Slika 27: Zgradba značke VLAN.

podobnih storitev, je tako ločevanje nujno.

Ločevanje posameznih vrst prometa v svoje VLAN-e olajšuje nadzor nad omrežjem in preprečuje širjenje napak iz enega v drugo.

IEEE 802.1Q označitev VLAN namenja 12 bitov identifikatorju VLAN-a (VLAN ID ali kratko VID). Tehnično je mogoče torej označiti 4096 različnih VLAN-ov. Ker sta 0 in 4095 že uporabljena (prvi za promet, ki v resnici ne nosi označbe VLAN, le prioriteto, drugi pa je za interno uporabo mostov in stikal), je v resnici na voljo 4094 navideznih omrežij.

Zgradba glave VLAN je prikazana na sliki 27.

8.3.2.3 Vrste glav VLAN

Pri nekaterih vrstah uporabe, na primer pri storitvi *en VLAN na odjemalca* (angl. *one VLAN per client*), je 4094 prehuda omejitev.

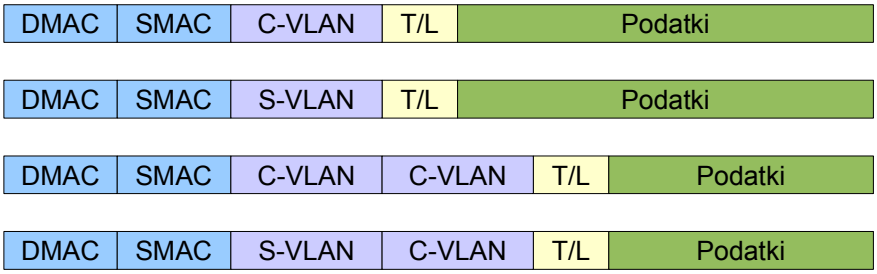
V takih primerih je mogoče gnezdenje glav VLAN. Tehnološko za globino gnezdenja ni drugih ovir kot velikost okvirja, ki ga narekuje prvi nivo. V tem primeru se mrežna oprema pri odločanju glede posredovanja paketa opira na najbolj zunanjo značko VLAN, omogoča pa dodajanje (angl. *tagging*) in odvzemanje (angl. *untagging*) oznak VLAN. V praksi je večinoma v uporabi enojno in dvojno označevanje okvirjev [7][9], ki sta tudi dobila svoji imeni – S (angl. *service*) in C (angl. *client*).

Format glave VLAN je v obeh primerih enak, lahko pa se razlikujeta po vrednosti polja `EtherType`. V uporabi so naslednje vrednosti:

- `0x8100` – v okviru IEEE 802.1Q standardizirani `EtherType` za C-VLAN.
- `0x88a8` – o okviru IEEE 802.1Q standardizirani `EtherType` za S-VLAN.
- `0x9100` – Cisco `EtherType` za S-VLAN.

V nekaterih primerih se tudi za S-VLAN uporablja `EtherType 0x8100`.

Poleg teh je mogoče predvsem za S-VLAN `EtherType` definirati poljubno vrednost – seveda na lastno odgovornost. Da je mera polna, je pri nekaterih mostovih mogoče nastaviti še spisek dodatnih kod, ki jih bo prepoznal kot S-VLAN `EtherType`, ne bo pa z njimi označeval odhodnih okvirjev.



Slika 28: Kombinacije označevanj VLAN, ki se pogosto uporabljajo.

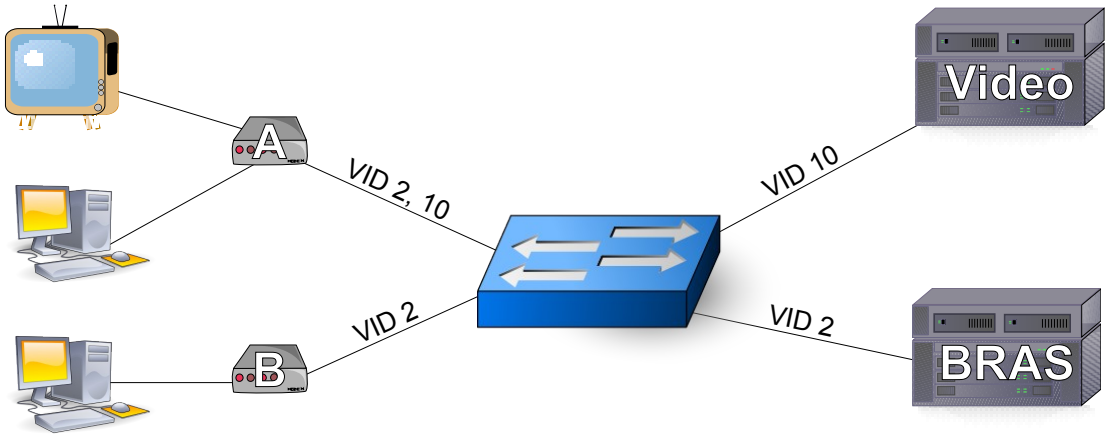
Na sliki 28 so prikazane vrste označevanj VLAN, ki se uporabljajo v sodobnih omrežjih. Izbira je odvisna od omrežja in storitev, ki jih ponudnik omrežja ponuja strankam.

8.3.2.4 Omejevanje dostopa

Odvisno od na vmesnik priklopljene mrežne opreme lahko sprejmemo različne okvirje: neoznačene, enojno ali dvojno označene. Včasih na istem vmesniku lahko sprejemamo vse tri vrste prometa, včasih dovolimo le označene okvirje, drugič le neoznačene.

Če dovolimo sprejem označenih okvirjev, potem lahko tudi bolj natančno določimo, katere VLAN-e bomo sprejemali. Tovrstno omejevanje dostopa ponuja veliko možnosti, ki jih operaterji s pridom izkoriščajo za nadzor nad dostopnostjo posameznih storitev.

Na sliki 29 vidimo omrežje, v katerem se VLAN 2 uporablja za dostop do interneta, VLAN 10 pa za dostop do internetne televizije. Vmesnik uporabnika A ima dostop do obeh VLAN-ov, zato uporabnik A lahko uporablja obe storitvi. Vmesnik uporabnika B ima dostop le do VLAN-a 2, zato ne more spremljati internetne televizije.



Slika 29: Omejevanje dostopa do storitev s pomočjo VLAN-ov.

V primeru postavitev, kot je ta na sliki 29, je na modemu določen vmesnik, preko katerega lahko spremljamo TV. Promet iz tega vmesnika se že na modemu označuje z vnaprej določenim VLAN-om, v našem primeru 10. Promet na ostalih vmesnikih se označuje z drugim VLAN-om, lahko pa tudi ostaja neoznačen in označevanje opravimo kasneje na mostu.

8.3.2.5 Prioriteta

Z uvedbo glave VLAN dobimo v okvirju Ethernet tudi prostor za hrambo prioritete. Trije biti omogočajo osem prioritetnih nivojev.

Z njihovo pomočjo lahko začnemo ločevati promet po prioriteti (lahko določimo prioriteto za vsak VLAN, lahko pa tudi znotraj posameznega VLAN-a). Lahko se spustimo še bolj v globino posameznega okvirja in prioriteto določimo tudi glede na njegovo vsebino.

Na podlagi prioritete lahko omejujemo količino prometa, ki smo jo pripravljeni sprejeti (angl. *policing*), lahko pa tudi oblikujemo odhodni prometni tok (angl. *traffic shaping*).

Možnost določanja prioritete odpira veliko novo področje kvalitete storitev (angl. *Quality of Service, QoS*, [33], [29]).

8.3.2.6 Odstranjevanje označitev

Končni uporabniki praviloma ne vedo ničesar o označevanju VLAN. Mehanizem je zanje povsem skrit. Deloma ga izvaja omrežna oprema v jedru omrežja (stikala in mostovi), zadnjo besedo pa imajo modemi, preko katerih so uporabniki priklopljeni na omrežje.

Nekateri modemi ne znajo izvajati označevanja VLAN in odstranjevanja označitev VLAN. Drugi modemi to sicer znajo, a skrbniki omrežja modemom ne zaupajo in zato izberejo, da se bo vse dodajanje in odstranjevanje označitev VLAN dogajalo na mostovih.

8.3.2.7 Prehajanje med VLAN-i

Pogosto most sprejme promet po enem VLAN-u (to pomeni, da ima dohodni promet v označitvi VLAN shranjeno določeno vrednost VLAN ID), oddajo pa želimo izvajati v drugem VLAN-u. Temu rečemo prehajanje med VLAN-i (angl. *VLAN remarking*, *VLAN translation*).

Naprednejše prehajanje med VLAN-i vključuje poleg spremembe VLAN ID-ja, tudi dodajanje in brisanje označitev VLAN.

8.3.3 Nadzor prometa

Nadzor prometa poteka v dveh korakih:

- ob sprejemu izvajamo krmiljenje,
- ob oddaji izvajamo glajenje.

V obeh primerih želimo promet približati vnaprej dogovorjenim mejam (angl. *Service Level Agreement*, SLA).

Krmiljenje in glajenje prometa močno zmanjšata možnost, da bo v omrežju prišlo do izgub zaradi zapolnitve medpomnilnikov. Krmiljenje je lažje za izvedbo, zato ga tudi pogosteje srečamo.

Krmiljenje in glajenje po drugi strani pomagata izboljšati izrabo obstoječe infrastrukture. Ker je pomnilnik, potreben za izvedbo čakalnih vrst, danes razmeroma poceni, gre za dokaj priljubljeno rešitev: večanje količine pomnilnika ima namreč neposreden vpliv na kvaliteto krmiljenja in glajenja.

8.3.3.1 Krmiljenje prometa

Krmiljenje prometa (angl. *traffic policing*) zagotavlja, da količina in vrsta prejetega prometa odgovarja vnaprejšnjemu dogovoru. Izvajamo ga ob sprejemu.

Promet najprej razvrstimo v prometne razrede (angl. *traffic class*), potem pa za vsakega od njih lahko predpišemo različne omejitve:

- vršna hitrost (angl. *Peak Cell Rate*, PCR) in
- trajna hitrost (angl. *Sustained Cell Rate*, SCR).

Vršna hitrost predpisuje, koliko prometa smo največ pripravljeni sprejeti ob kratkih izbruhih. Presežni promet odmetavamo.

Trajna hitrost je manjša ali kvečjemu enaka vršni hitrosti. Gre za količino prometa, ki smo jo pripravljeni sprejemati dlje časa. Vmes lahko za kratek čas količina prometa tudi preseže trajno hitrost, a temu mora slej ko prej slediti obdobje, ko bo prometa manj, kot je predpisana trajna hitrost.

8.3.3.2 Glajenje prometa

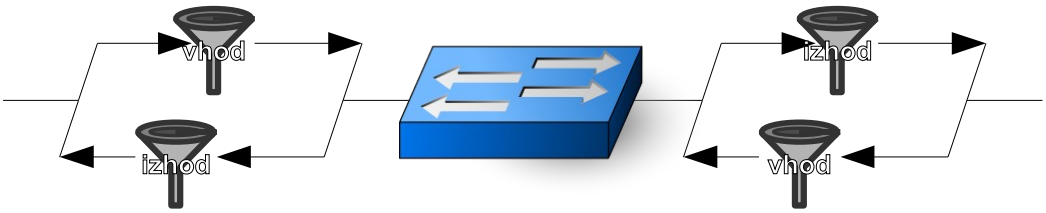
Tudi glajenje prometa deluje na podlagi vršne in trajne hitrosti, ob tem pa je pomembna še ena vrednost – velikost medpomnilnika.

Gladilnik prometa (angl. *traffic shaper*) ob oddaji poskrbi, da nikoli ne presežemo vnaprej določene meje. Če mu v oddajo predamo preveč prometa, ga začasno shrani v medpomnilniku, kasneje pa ga prepušča v skladu z dogovorom. Od velikost medpomnilnika je odvisno, kako velike izbruhe bo gladilnik še znal pokriti brez izgube prometa[29].

8.3.4 Filtriranje

Zgodovinsko je bil drugi nivo omrežja deležen zaupanja, požarne pregrade pa so se ukvarjale z vdori šele na višjih nivojih. Glavni razlog za to je omejen doseg napadov na drugem nivoju. Z rastjo omrežij L2, predvsem imamo v mislih omrežja ponudnikov dostopa do interneta, je tudi varnostni pogled postal pomembnejši. Danes noben resen most ne more izpustiti tega področja [30].

Le redki mostovi ne podpirajo kake vrste filtriranja okvirjev. Od obravnavanih je taka le Linuxova izvedba mosta, ki pa to rešuje drugače. Pri nekaterih gre le za izločanje določenih tipov okvirjev, pri drugih pa so filtri podlaga za kakšno posebno vrsto obdelave. Na vsak način lahko funkcijo filtriranja ločimo na dva sklopa: izbira in akcije.



Slika 30: Most z dodanimi filtri.

29 Če izbruh traja predolgo, se medpomnilnik zapolni. V tem primeru pride do izgube prometa.

Podjetje Cisco, posledično pa tudi večina ostalih proizvajalcev mrežne opreme, za filtriranje uporablja izraz *kontrola pristopa* (angl. *access control*, ACL).

Ločimo vstopne (angl. *ingress*) in izstopne (angl. *egress*) filtre. Prvi delujejo na vhodnem, drugi pa na odhodnem prometu (slika 30).

8.3.4.1 Izbira

Ker so mostovi naprave L2, je najbolj običajna podpora izbirnih pogojev na podlagi polj L1 in L2 (to vključuje še glave VLAN). Redkeje je podprto tudi filtriranje na podlagi polj L3 ali L4 (angl. *Deep Packet Inspection*, DPI) – to je prepuščeno usmerjevalnikom in posebej temu namenjenim napravam.

Pri filtriranju na podlagi podatkov 3. in 4. nivoja se moramo zavedati tudi etičnih in pravnih strani takega početja. V primeru paketov IPv4 in IPv6 nam vpogled v podatke 3. in 4. nivoja lahko posreduje precej informacij o naravi aktivnosti uporabnika ([42]). Kdo, kaj, kako, za kakšen namen in koliko časa sme zbirati, obdelovati in hraniti tovrstne podatke, je domena zakonodaje posameznih držav, med njimi pa so velike razlike.

8.3.4.2 Akcije

Najosnovnejša akcija, ki jo v zvezi s filtriranjem podpirajo mostovi, je blokiranje. S tem preprečimo vdor nezaželenih okvirjev v omrežje, posledično pa je omrežje bolj varno. Prav tako na ta način onemogočimo uporabnikom dostop do storitev, do katerih niso upravičeni (recimo do IP TV v primerih, ko je nekdo le naročnik podatkovnega dostopa).

Že pri omejevanju dostopa se srečamo s težavo, da je včasih lažje določiti, kaj je dovoljeno, drugič pa, kaj je prepovedano. Zaradi tega je med akcijami poleg blokiranja običajno tudi prepuščanje. To omogoča, da s filtri določimo, da na nekem vmesniku prepuščamo promet IP do določenega naslova IP, vse ostalo pa blokiramo.

V večini držav imajo policije in tajne službe zakonsko podlago za izvajanje prisluhov v omrežjih. Izvajanje takega prisluha na celotnem omrežju je težko izvedljivo (pa tudi zakonsko težko opravičljivo), za posameznega uporabnika (ponavadi to pomeni za posamezen vmesnik) pa povsem izvedljivo. V takem primeru promet, ki je namenjen na ali pa prihaja iz tega vmesnika, zrcalimo na nek tretji vmesnik, tam pa je priklopljen analizator prometa.

8.3.5 Televizija in telefonija

Uvedba videa, zaenkrat predvsem internetne televizije (angl. *IP TV*), predstavlja za omrežja dodaten izziv, saj je treba dostaviti pravi videotok do pravih naročnikov. Dostava videa nenaročenim ne pomeni le izgube dohodka za operaterja, pač pa tudi nepotrebno zapravljanje omejene pasovne širine za naročnika in operaterja. Odgovor na to je skupinsko pošiljanje [18]. Gre za kombinacijo protokolov L2 in L3 [10], ki poskrbijo, da gre pravi video k pravim naročnikom. Seveda je povsem isto mogoče uporabljati tudi za distribucijo zvoka.

Tudi internetna telefonija (angl. *Voice over IP*, VoIP) predstavlja poseben izziv. Zahtevana pasovna širina tukaj ni tako problematična. Glavno težavo predstavljajo zakasnitve. Te pri govoru opazimo dosti hitreje kot pri sliki, poleg tega pa gre pri telefonskem pogovoru za dvosmerno komunikacijo, zato te zamude dosti bolj opazimo kot pri gledanju televizije. Glavni odgovor na to predstavlja uvedba različnih tehnologij za zagotavljanje kakovosti storitev (angl. *QoS*).

8.3.5.1 Skupinsko pošiljanje

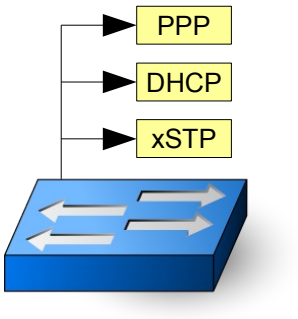
Pri samodejnem učenju tabele naslovov MAC smo omenili, da se most ne uči naslovov, ki imajo vklopljen bit za skupinsko pošiljanje in pošiljanje na vse naslove. Tako se samodejno ne nauči nobenega skupinskega naslova[30]. Če most ne ve, kateri vmesniki spadajo v posamezno skupino, potem mu kot edina izbira ostane pošiljanje vsega skupinskega prometa na vse vmesnike. V primeru televizije to preprosto ni mogoče, saj je pasovne širine premalo.

Zaradi tega mostu pomaga zunanja komponenta, *IGMP vohun* (angl. *IGMP Snooper*), ki spremlja pakete IGMP in se na podlagi tega nauči, kdo je prijavljen v katero skupino. Podrobnosti delovanja protokola IGMP nas na tem mestu ne zanimajo. Za nas je pomembno le, da neka entiteta v sistemu spremlja pakete IGMP in nam na njihovi podlagi sporoča, kateri vmesniki spadajo v katero skupino.

Ko most sprejme okvir, za katerega na podlagi ciljnega naslova MAC ugotovi, v katero skupino za skupinsko pošiljanje spada, posreduje okvir vsem vmesnikom, ki spadajo v to skupino[31].

8.3.6 Podpora aplikacijskim protokolom

Stikala omogočajo sodelovanje z različnimi protokoli, ki so odvisni od posamezne postavitve. Operaterji tako uporabljajo protokol DHCP za dostavo osnovnih omrežnih nastavitev, protokol PPP za povezovanje, protokol IGMP za upravljanje internetnega videa in radia, protokol OAM za preverjanje stanja povezav na različnih nivojih [8] itd.



Slika 31: Most in aplikacijski protokoli.

To so protokoli, ki so prezahtevni, da bi sodili v jedro stikal in mostov. Poleg tega se ves čas pojavljajo novi. Zaradi tega je pomembna zahteva, ki jo mora izpolnjevati vsak spodoben most, da omogoči vključevanje tovrstnih razširitev (slika 31).

8.3.6.1 Kaj je aplikacijski protokol

Ne moremo mimo vprašanja, kdaj je nek protokol aplikacijski in kdaj ne. S tem izrazom označujemo protokole, ki niso nujni za delovanje mosta, vendar pa na nek načni tesno z njim sodelujejo:

30 V resnici skupinski naslov tudi nikoli ne sme biti izvorni naslov okvirja in se ga tudi zato ne bi mogli naučiti na tak način.

31 Posebna razširitev lahko poleg ciljnega naslova MAC upošteva tudi ciljni naslov IP. Na ta način je izraba pasovne širine zares optimalna, saj je izključeno prekrivanje po 32 skupinskih naslovov IP v enem skupinskem naslovu MAC. Tako delovanje pa presega L2, na katerem bi naj deloval most.

- Protokol xSTP prekinja krožne povezave, ki lahko nastanejo pri povezovanju Ethernetovih omrežij. Most brez tega protokola deluje, a lahko po nepotrebnem trati pasovno širino.
- Protokola PPP in DHCP nimata vpliva na delo mosta, potrebujeta pa podatke, ki jih lahko preskrbi most (vhodni vmesnik in VLAN).
- Protokol IGMP sicer na most vpliva, a bi bilo delovanje mogoče tudi brez njega. IGMP mostu omogoča, da lahko bolje izrablja vire (pasovno širino).
- Protokol CFM (angl. *Connectivity Fault Management*, [8]) tesno sodeluje z mostom. Sicer ne vpliva na delovanje mosta, potrebuje pa od njega mnoge podatke. Poleg tega mora biti za pravilno delovanje zelo tesno povezan z mostom.
- Protokol Ethernet OAM, znan tudi kot IEEE 802.3, določba 57 (angl. *Clause 57*, [14]), ne sodeluje z mostom, ga je pa treba umestiti med most in vmesnik.

Okvirje, ki pripadajo aplikacijskim protokolom, znamo opisati na tak način, da jih lahko potem med delovanjem „lovimo“ med ostalim prometom. To lahko počnemo kar s filtri, lahko pa je temu namenjen tudi kak poseben procesni blok.

IGMP okvirje lahko opišemo kot okvirje IPv4 ali IPv6 (torej imajo `EtherType 0x0800` ali `0x86dd`), vsebovani protokol IP pa ima kodo `0x02`:

```
IGMPFrame =
(Ethernet.EtherType == 0x0800 AND IPv4.Protocol == 0x02)
OR
(Ethernet.EtherType == 0x86dd AND IPv6.NextHeader == 0x02)
```

8.3.7 Zaščita pred preobremenitvijo in zlorabami

Od mosta pričakujemo, da deluje pravilno, kadar na vhodu dobiva pravilen promet v zmernih količinah. Se pa na to pri omrežni opremi ne moremo zanašati. Napake in namerne zlorabe lahko pripeljejo do okvarjenega prometa. V takih primerih se mora most pravilno odzvati. Prevelika količina prometa je prav tako problematična. V najboljšem primeru je posledica le oslabljeno delovanje mosta za enega uporabnika, v najslabšem pa lahko pomeni tudi prenehanje delovanja celotnega mosta, to pa ima posledice za vse uporabnike.

Omenili smo že, da je včasih za drugi omrežni nivo veljalo popolno zaupanje, danes pa ni več tako.

V zvezi z obremenitvami obstaja zelo preprost način napada, ki vključuje posege na drugem nivoju. Za prekinjanje zank se uporablja protokol STP. Njegovo delovanje je odvisno od podatkov, ki si jih izmenjujejo mostovi v obliki paketov BPDU (angl. *Bridge Protocol Data Unit*). Z njihovo pomočjo se sodelujoči mostovi dogovorijo za vzpostavitev topologije. S potvorjenimi paketi BPDU lahko dosežemo, da naš most vsaj začasno velja za korenski most (angl. *root bridge*) [30]. V takem primeru lahko pričakujemo naval prometa, na katerega moramo ustrezno reagirati.

8.3.7.1 Preobremenitev CPU

Predvsem za mostove je zelo pomembno, da ščitijo same sebe pred preobremenitvijo. S tem imamo v mislih položaj, ki nastopi zaradi pravilne ali nepravilne uporabe, ko povpraševanje

po CPU preseže dostopne vire.

- Sprejemljivo je, da po prioritetnem redu pada kakovost posameznih storitev.
- Manj sprejemljiv je splošen naključen upad kvalitete storitev.
- Povsem nesprejemljiva pa je neodzivnost upravljalnega vmesnika.

Pri programskih rešitvah je na voljo več mehanizmov, s katerimi most poskrbi, da ne pride do stradanja drugih delov sistema:

- Lahko omeji največji delež CPU, ki ga bo porabil.
- Lahko zagotovi minimalno pogostnost prepuščanja CPU drugim.
- Lahko v sodelovanju z mehanizmi za nadzor porabe virov (angl. *watch dog*) dinamično spreminja največjo dopustno obremenitev.
- Lahko ugotavlja poskuse zlorab na posameznih vmesnikih in jih začasno onemogoči.
- Lahko določi maksimalno prepustnost posameznega vmesnika, največjo količino prometa za določen VLAN itd.

Glavna težava pri vseh teh mehanizmih je, da ne želimo biti preveč previdni in posledično vplivati na dopustno obnašanje, hkrati pa ne preveč popustljivi, da ne bi s tem povzročili pretirane škode sebi in preostanku omrežja.

8.3.7.2 Zloraba pasovne širine

Zloraba pasovne širine je podoben problem kot preobremenitev CPU, le da je tokrat cilj pasovna širina. Tudi metode boja proti tem zlorabam so podobne, je pa večji poudarek na omejevanju zlorab.

Najbolj pogost način zlorabe je izkoriščanje neuspešne poizvedbe v tabeli naslovov MAC (DLF). V tem primeru se okvir razpošlje vsem vmesnikom (vsem, ki pripadajo VLAN-u). Določeno število neuspešnih poizvedb je normalno in sprejemljivo, če jih je preveč, pa to kaže na mogoč poskus zlorabe[32].

8.3.7.3 Razlika v primerjavi s prioriteto

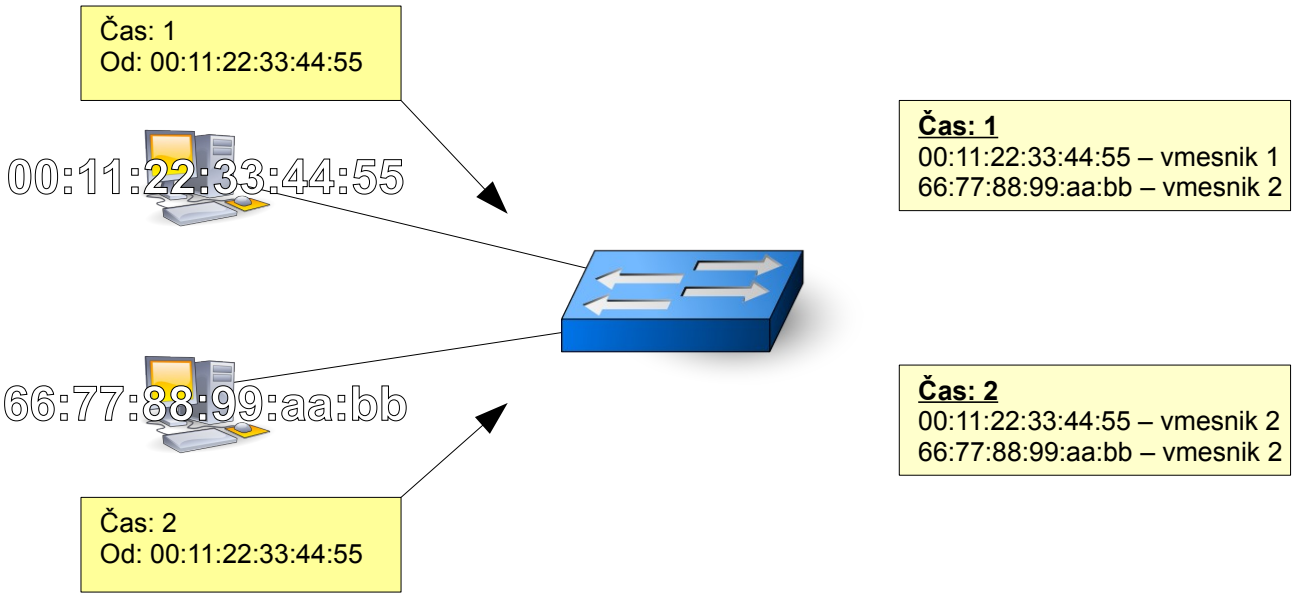
Zaščita pred preobremenitvijo in zlorabami zveni podobno kot obravnava prometa glede na prioriteto. To drži, z eno pomembno razliko: prioriteto obravnavamo šele potem, ko smo s predhodno zaščito zagotovili, da si to lahko privoščimo.

Izvedbeno je razlika še bolj preprosta: zaščita pred preobremenitvijo je med prvimi stvarmi, ki jih most preverja, ko začne obdelovati nov okvir. Če ugotovi prekoračitve, je tako prihranek večji. Obravnavanje prioritete nastopa kasneje, posledično pa je tudi cena, ki jo do takrat plačamo v obliki porabljenih sistemskih virov, višja.

32 Vsak porast DLF ne pomeni takoj zlorabe. Tudi napake pri nastavitvah se lahko demonstrirajo kot DLF.

8.3.7.4 Potvorjen promet

Posredovanje prometa je odvisno od naučenih naslovov MAC. To pomeni, da lahko s potvarjanjem naslova MAC pridemo do okvirjev, ki bi sicer morali biti poslani drugam. Gre za varnostno tveganje, zato to poskušamo preprečiti. Funkcionalnost se imenuje varovanje vira MAC (angl. *MAC Source Guard*), gre pa za to, da ne dovolimo učenja že naučenega naslova MAC na drugem vmesniku. Naslov se na drug vmesnik lahko preseli šele, ko na prvem zastara. Tako je prehajanje naprav med naročniškimi vmesniki še vedno mogoče.



Slika 32: Potvarjanje naslova MAC.

Na sliki 32 vidimo primer, ko računalnik z naslovom MAC `66:77:88:99:aa:bb` laže. Ko pošlje okvir, v katerem piše, da ga je poslal `00:11:22:33:44:55`, s tem most prepriča, naj ves promet za ta naslov MAC pošilja na njegov vmesnik. Tako bo ostalo, dokler prvi računalnik ne bo poslal vsaj enega okvirja.

Težava obstaja tudi nivo višje. Mnogi napadi na računalniška omrežja sebe poskušajo zakriti tako, da se se prikazujejo, kot da izvirajo od nekod drugod. Računalnik z naslovom IPv4 `1.1.1.1` lahko brez težav pošlje paket, v katerem laže, da paket v resnici prihaja iz naslova `2.2.2.2`. Za koristne komunikacije to ni uporabno, saj bo prejemnik odgovor poslal napačnemu naslovniku. Je pa to zelo priročno za zlonamerne napadalce, ki tako zabrišejo sled za sabo.

Podobno kot smo prej preprečevali selitev naslova MAC na drug vmesnik, zdaj potrebujemo način preprečevanja pošiljanja z neveljavnih naslovov IPv4[33]. Gre za varovanje vira IP (angl. *IP Source Guard*). Prepuščamo le pakete, ki prihajajo iz določenih naslovov IPv4 – tistih, ki so bili ročno dodani, in tistih, ki so bili naučeni s prisluškovanjem protokolu DHCP.

Res pravo rešitev, ki preprečuje potvarjanje prometa, omogočajo bolj zapleteni protokoli [19].

33 Varovanje naslovov IP očitno sodi na tretji nivo, ne na drugega, a je to vseeno vedno bolj pogosta funkcija mosta.

8.3.7.5 Okvarjen promet

Del okvarjenega prometa zavračajo že nižji nivoji. Tako nivo Ethernet na podlagi kontrolne vsote že zavrača pakete, ki so se okvarili med prenosom. To pa ne pomeni zaščite pred paketi, ki so bili že poslani okvarjeni. Od mosta pričakujemo, da bo take pakete zaznal in zavračal.

Na srečo so protokoli, s katerimi ima opravka most, razmeroma preprosti s stališča razčlenjevanja okvirjev (angl. *frame parsing*). Pri tem imamo v mislih dejstvo, da imajo glava Ethernet in dodatne označbe, ki jih most še upošteva, fiksno velikost.

Višji protokoli pa niso več taki. Glava IPv4 vsebuje polje, v katerem je zapisana dolžina (angl. *header length*) glave z dodatnimi opcijami. Polje je 4-bitno in vsebuje velikost glave IPv4 v 4-zlogovnih korakih. Največja vrednost v polju je 15, največja velikost glave IPv4 pa 60 zlogov. To pomeni, da pri največji vrednosti polja prvi zlog znotraj IPv4 vsebovanega protokola (TCP, UDP ...) že presega najmanjšo velikost okvirja Ethernet. Če ob razčlenjevanju vsebine poleg vrednosti polj v posameznih glavah ne preverjamo tudi dolžine okvirja, potem je možnost za napačno delovanje precejšnja.

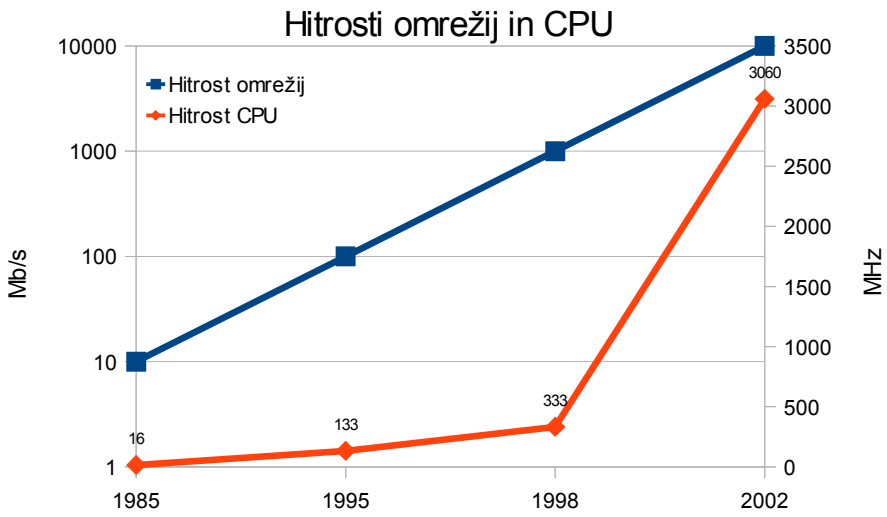
Področje raziskovanja odziva omrežne opreme na okvarjen promet je razvejano in nadvse živahno, zato se mu v tem delu ne bomo podrobno posvečali.

8.3.8 Visoko odzivna omrežja

Odzivnost omrežja je rezultat hitrosti prenosa in odzivnosti stikal. Ob tem, ko hitrosti povezav zrastejo vsakih nekaj let, je privzeta predpostavka, da temu sledi tudi hitrost omrežne opreme. Pogledali bomo, kako so sestavljene zakasnitve v omrežju in kako napredna stikala te zakasnitve zmanjšajo na najmanjšo možno mero.

Na sliki 33 je prikazana rast hitrosti omrežij in CPU[34]. Opazimo lahko, da so v času, ko so omrežja naredila preskok med 10 in 100 Mb/s, podoben hitrostni preskok naredili tudi procesorji (v resnici nekaj manjši). V času, ko so omrežja pospešila za faktor 100, so procesorji pospešili le za faktor 20 (leto 1998), kasneje pa so procesorji sicer začeli loviti zaostanek, a ga še zdaleč niso ujeli.

34 Pri CPU je kot merilo uporabljena hitrost sistemske ure, ki pa prave hitrosti ne odraža najbolje. Ta v večini primerov zaostaja za hitrostjo sistemske ure.



Slika 33: Rast hitrosti omrežij in CPU ([45]).

Pri tem je treba poudariti, da se je nekdanje težišče razvoja procesorjev odmaknilo od tekme za čim višjo hitrost sistemske ure. Današnji procesorji imajo več jeder in dodatne funkcijske enote, ki so posvečene določenim pogostim opravilom. Primerjava sistemskih ur je zato vedno bolj neprimerna.

Če je naš cilj visoko odzivno omrežje, potem moramo svoj trud usmeriti v zmanjševanje zakasnitev in pospeševanje obdelave okvirjev.

Najprej bomo zato pogledali, na kakšne vrste zamud pri delu z okvirji naletimo, sledi groba ocena teh zamud, za konec pa bomo pogledali tri strategije za zmanjšanje zamud.

8.3.8.1 Zamude pri delu z okvirji

Celotno pot okvirja od pošiljatelja do naslovnika lahko časovno razdelimo na:

- prenos – čas, porabljen za prenašanje bitov preko različnih medijev;
- obdelava – čas, porabljen pri delu z okvirjem na vseh stikalih in usmerjevalnikih na poti;
- čakanje – čas, porabljen za čakanje v različnih vmesnih pomnilnikih.

Končni čas od pošiljatelja do prejemnika je njihova vsota, cilj visoko odzivnega sistema pa, da to vsoto kar najbolj zmanjša.

Pri krajšanju prenosnih časov smo omejeni z napredkom tehnologije, ta pa nenazadnje vedno bolj zadeva tudi ob fizikalne omejitve. Tukaj torej velikih pospeškov ne moremo iskati.

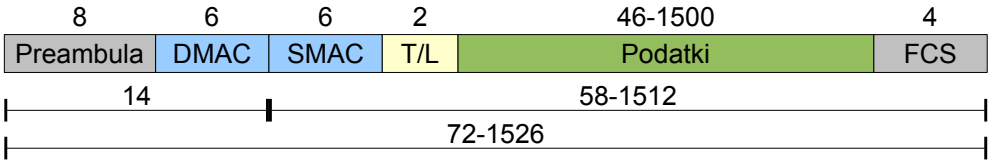
Tudi pri krajšanju časa obdelave smo odvisni od napredka tehnologije, poleg tega pa lahko pridobimo tudi z uporabo hitrejših algoritmov. Še najbolj potratno je iskanje po usmerjevalnih tabelah in zato so izboljšave tega dela tudi cilj dela mnogih raziskovalcev.

Krajšanje časa, porabljenega v medpomnilnikih, v resnici ni odvisno od tehnološkega napredka, pač pa predvsem od organizacije obdelave okvirjev. Delo z okvirji lahko razdelimo na korake. Če med njimi okvirje odlagamo v medpomnilnike, potem celotna obdelava lahko

traja dosti dlje.

8.3.8.2 Časovna ocena zamud

Vrnimo se za nekaj časa na zgradbo okvirja Ethernet (slika 34). Minimalno količino podatkov, potrebno za usmerjanje, stikalo dobi že s prejemom ciljnega naslova MAC. Torej že po prebranih 14 zlogih (preambula in ciljni naslov MAC) oziroma 112 bitih. Celoten okvir lahko doseže velikost 1.526 zlogov oziroma 12.208 bitov, zagotovo pa bomo sprejeli vsaj 72 zlogov oziroma 576 bitov.



Slika 34: Najmanjše potrebno in celotno število prejetih zlogov pred začetkom dela z okvirjem.

To je pomembno zaradi celotnega zamika, ki ga povzroči obdelava okvirja v stikalu. Potrebne podatke imamo že zelo hitro, v najslabšem primeru pa bomo najprej prebrali celoten okvir in šele potem začeli z obdelavo. Koliko časa preteče vmes, je odvisno od hitrosti fizične povezave (formula 1).

$$\delta\,\%t = \frac{data}{S_{line}} \tag{1}$$

V fiksnih omrežjih sta danes najbolj pogosti hitrosti 100 Mb/s in 1 Gb/s. V primeru 100 Mb/s to pomeni, da sprejem celotnega okvirja polne dolžine (1500 zlogov podatkov) povzroči časovni zamik 0,00012096 sekunde. Na prvi pogled ni veliko, a zamiki se seštevajo. Če okvir od uporabnika do strežnika preči 10 stikal, potem časovni zamik naraste že na 0,0012 sekunde. Tudi če strežnik odgovori takoj (kar ni verjetno), bo zamud do takrat, ko bo uporabnik prejel odgovor, že 0,0024 sekunde. Pri tem gre za zamudo, ki ni posledica nikakršne obdelave, ampak izhaja le iz načina dela s paketom.

$$\begin{aligned} S_{line} &= 100\,Mbit/s \\ S_{CPU} &= 500\,Mops/s \end{aligned} \tag{2}$$

Tem številkam moramo nujno postaviti ob bok druge – operacije, ki jih moderni procesorji zmorejo v tem času. Uporabili bomo zelo površno oceno. Recimo, da imamo opravka s procesorjem s sistemskih taktom 1 GHz, poleg tega pa predvidimo še, da je polovica časa porabljena za čakanje zaradi zgrešitev v predpomnilniku. To še vedno pomeni, da procesor v sekundi izvede 500.000.000 operacij.

$$N_{ops} = \delta\, t \cdot S_{CPU} \tag{3}$$

$$data_{min} = 58\,bytes = 464\,bits$$

$$N_{ops_{min}} = \frac{464\,bits}{100\,Mbit/s} \cdot 500\,Mops/s = 2320\,ops \tag{4}$$

$$data_{max} = 1512\,bytes = 12096\,bits$$
$$N_{ops_{max}} = \frac{12096\,bits}{100\,Mbit/s} \cdot 500\,Mops/s = 60480\,ops \quad (5)$$

V primeru 100 Mb/s hitrosti omrežja lahko tak procesor od prejema potrebnih podatkov za nadaljnjo obdelavo (prebranih 14 zlogov) pa do prejema celotnega okvira (še naslednjih 58-1512 zlogov) opravi od 2.320 do 60.480 operacij (formuli 4 in 5). V primeru kratkih okvirjev je prihranek majhen, v primeru velikih okvirjev pa ocenjujemo, da število operacij CPU zadostuje za celotno obdelavo okvirja.

8.3.8.3 Načini dela z okvirji

Odvisno od načina posredovanja posameznega okvirja stikala in mostovi delujejo v enem od naslednjih načinov ([25], [36], [39]):

- *Pretočni (angl. Cut-Through*) – Posredovanje okvirja se začne takoj po prejemu in poizvedbi po ciljnem naslovu MAC. Časovni zamik je tako manjši, saj se poizvedba lahko začne, še preden je sprejet celotni okvir.
- *Shrani in posreduj (angl. Store-And-Forward*) – Okvir je sprejet v celoti, šele potem se izvede poizvedba po ciljnem naslovu MAC. Pred posredovanjem lahko preverimo pravilnost celotnega okvirja in izločimo tiste okvirje, v katerih odkrijemo napake.
- Pretočni brez ostankov (angl. *Fragment-Free*) – Različica pretočnega načina, pri kateri stikalo pred nadaljnjo obdelavo sprejme prvih 64 zlogov okvirja. To rešuje problem poznih trkov (angl. *late-collision problem*).

Vedno pogostejše so tudi izvedbe pretočnih stikal, ki pa poleg ciljnega naslova MAC preverjajo tudi druga polja. Posredovanje se ustrezno zakasni, je pa na ta način omogočeno upoštevanje pestrejšega nabora informacij.

Pretočna izvedba zahteva zelo tesno sodelovanje med fizičnim (L1) in povezovalnim nivojem (L2). Zaradi tega pretočni način ni primerna izbira, kadar želimo arhitekturno čisto izvedbo stikala ali pa imamo L1 izveden strojno, L2 pa programsko.

8.4 Notranja zgradba mosta

Čeprav so konkretne izvedbe mostov lahko precej različne, pa brez nekaterih skupnih elementov ne morejo.

Koraki obdelave okvirja v mostu, ki niso vezani na konkretno izvedbo, so:

- okvirje sprejemamo od omrežnega vmesnika;
- ob sprejemu okvirja opravimo vhodno obdelavo;
- na podlagi tabele naslovov MAC se odločimo, ali bomo okvir poslali na enega ali več izhodnih vmesnikov;
- pred oddajo opravimo izhodno obdelavo;

- okvir predamo mrežnemu vmesniku.

Iz tega lahko izluščimo naslednje komponente:

- omrežni vmesnik,
- tabela naslovov MAC in
- vhodno-izhodna obdelava.

8.4.1 Omrežni vmesnik

Izvedba omrežnega vmesnika (angl. *Internal Sublayer Service* oziroma ISS v izrazoslovju IEEE,[5],[6],[7],[8]) ni predmet tega dela. Od njega pričakujemo naslednje:

- sprejema okvirje Ethernet in nam jih predaja v obdelavo.
- Ko mu predamo okvir, ga preko L1 odda ostalim napravam na mrežnem segmentu.
- Lahko ima notranje medpomnilnike, vendar nas njihovo število in obnašanje ne zanima.
- V osnovi gre za vmesnik do L1.

8.4.2 Tabela naslovov MAC

Tabela naslovov MAC je katalog naslovov MAC, ki smo se jih naučili na posameznih vmesnikih. Podpirati mora samodejno časovno zastaranje vnosov, se učiti naslove MAC, po potrebi pa onemogoča učenje že naučenega naslova MAC na drugem vmesniku (premik iz enega vmesnika na drugega).

Velikost tabele je pomembna za dobro delovanje celotnega mosta ali stikala. Če je premajhna, potem bo z večjo verjetnostjo prišlo do primera, ko iskanega naslova MAC ne bo več v tabeli. Vzrok za to ne bo, da se ga nismo mogli naučiti, ampak da nismo imeli zadosti prostora. V takem primeru okvir pošljemo vsem vmesnikom, to pa je potratno tako s stališča mosta kot tudi s stališča porabe pasovne širine posameznih povezav. Prevelika tabela pa pomeni večjo porabo pomnilnika in zato lahko tudi počasnejše iskanje.

```
PortList ports mac_table_lookup( MacAddress destination_mac )
{
     MacTableEntry entry = mac_table_find(destination_mac);

     if ( is_multicast_address(destination_mac) ) {
          if ( entry )
               return entry.port_list; // List of multicast ports.
          else
               return NULL; // No one registered for this group.
     }
     else {
          if ( entry )
               return entry.port; // There is one port.
          else
               return all_ports;
     }
}
```

Slika 35: Psevdokoda poizvedbe v tabeli naslovov MAC.

Postopek iskanja v tabeli naslovov MAC je prikazan na sliki 35.

Zaradi varnosti potrebujemo tudi način, da omejimo število vnosov naslovov MAC. Brez tega bi zlonamerni uporabnik na enem vmesniku lahko zasedel celotno tabelo in tako povzročil, da bi se ves ostali promet razpošiljal (angl. *flood-forward*, poplavljanje) vsem vmesnikom. Posledici bi bili zasičenost linij in nezaželeno prepuščanje prometa, namenjenega k enemu uporabniku, vsem drugim.

Tabela naslovov MAC je poleg naštetega tudi najbolj primerno mesto za podporo skupinskega pošiljanja. Pravzaprav se od običajnih vnosi za skupinsko pošiljanje razlikujejo le po tem, da je lahko naslov za skupinsko pošiljanje naučen na več vmesnikih[35].

8.4.2.1 Tabela naslovov MAC in podpora za označevanje VLAN

V prvih stikalih je bila tabela naslovov MAC taka, kot smo jo predstavili doslej. Tisti hip, ko začnemo bolj izdatno uporabljati označevanje VLAN, pa je smiselna razširitev tabele, da vanjo vključimo tudi to. Če pri posredovanju označbe VLAN zanemarimo, potem tvegamo posredovanje paketov tudi tja, kamor jih v najboljšem primeru ne bi bilo treba – gre za nesmotrno porabo pasovne širine, v najslabšem pa, kamor jih ne bi smeli – v tem primeru gre za varnostno luknjo.

Označevanje VLAN lahko podpremo na dva načina:

- Razširimo tabelo naslovov MAC, da pri poizvedbi poleg naslova MAC uporablja tudi VLAN ID.
- Uporabimo po eno običajno tabelo za vsakega od VLAN-ov.

Prva rešitev zahteva nekaj več dela, obenem pa ponuja možnosti za optimiziranje, predvsem glede porabe pomnilnika. Postopek poizvedbe v tabeli naslovov MAC, ki vključuje podporo za označbe VLAN, prikazuje slika 36.

35 V tem primeru *naučen* pomeni, da je na tem vmesniku nekdo prijavljen v skupino.

```
PortList ports mac_table_lookup(MacAddress destination_mac,
     VlanID vid)
{
     MacTableEntry entry = mac_table_find(destination_mac, vid);

     if ( is_multicast_address(destination_mac) ) {
          if ( entry )
               return entry.port_list; // List of multicast ports.
          else
               return NULL; // No such multicast group.
     }
     else {
          if ( entry )
               return entry.port; // There is one port.
          else
               return all_ports;
     }
}
```

Slika 36: Psevdokoda poizvedbe v tabeli naslovov MAC, ki podpira označbe VLAN.

8.4.3 Vhodna in izhodna obdelava

Vhodna in izhodna obdelava lahko potekata na več načinov. Najbolj splošno lahko celotno področje pokrijemo s pravili `if-then`, ki so vezana na posamezne vmesnike. Pravila se izvajajo v predpisanem vrstnem redu, dokler izvajanje ne naleti na konec spiska pravil, ali katero od akcij, ki prekinejo nadaljnje izvajanje. To sta akciji zavrzi (angl. *drop*) in posreduj naprej (angl. *pass*). Psevdokoda take vhodno-izhodne obdelave je predstavljena na sliki 37.

V praksi takega pristopa praktično ne srečamo. Običajno si v okviru vhodne in izhodne obdelave sledijo moduli, ki pokrivajo posamezne funkcionalne sklope mosta (označevanje VLAN, zagotavljanje kvalitete storitev, filtriranje, obdelava aplikacijskih protokolov itd.). Taka izvedba po eni strani ponuja dobre možnosti za lokalne optimizacije znotraj sklopov, po drugi pa vodi v podvajanje nekaterih funkcionalnosti.

Predvsem strojne izvedbe pogosto uporabljajo pristop, ki je do neke mere podoben obdelavi strojnih ukazov v cevovodu mikroprocesorja:

- najprej s pomočjo filtrov okvir dobi označbe, ki predpisujejo, kakšna naj bo nadaljnja obdelava (odmetavanje okvirja, dodajanje in brisanje označb VLAN, posredovanje na različne vmesnike, uvrščanje v določene izhodne vrste ...);
- drugi korak predstavlja izvajanje postopkov, ki so bili izbrani v prvem koraku;
- okvir na koncu pošljemo preko izbranih vmesnikov ali pa ga zavržemo.

Tak pristop je očitno zelo priročen za strojno izvedbo, vedno večje število jeder v modernih mikroprocesorjih, pa ga bo mogoče naredilo bolj priljubljenega tudi med programskimi izvedbami.

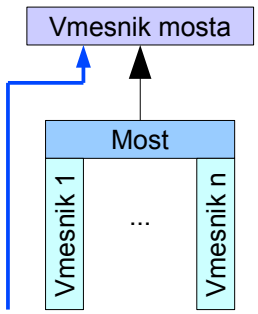
```
Frame process_frame(Frame frame, Port port, Direction dir)
{
     Bool stop = FALSE;
     ProcessingListEntry entry;
     ProcessingList proc_list = get_port_processing_list(port,
     dir);

     for each ( entry in proc_list ) {
          if ( check_condition(entry) )
               (frame, stop) = apply_actions(entry, frame);
          if ( frame == NULL || stop == TRUE )
               break;
     }

     return frame;
}
```

Slika 37: Psevdokoda vhodno-izhodne obdelave.

8.4.4 Upravljanje

Omenili smo že dve možnosti upravljanja (z uporabo fizično ločenih ali souporabo obstoječih vmesnikov). Če izberemo možnost, pri kateri se upravljalni in ostali promet mešata, potem je očitno, da ju moramo na nek način ločiti in upravljalni promet preusmeriti proti ustrezni upravljalni aplikaciji.

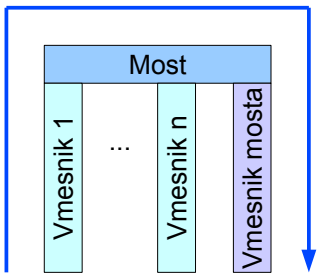
Standard IEEE 802.1D [5], ki opisuje funkcionalnost in zgradbo mostov, nakazuje možnosti za izvedbo ločenega upravljanja, a tega izrecno ne predpisuje. Mnoge izvedbe vseeno izberejo predlagano rešitev. Shematsko je ta prikazana na sliki 38 – upravljavski promet predstavlja modra puščica.



Slika 38: Shematski prikaz ene od možnosti za ločevanje upravljavskega prometa.

Prikazani način zahteva posebno obravnavo upravljavskega prometa in tako povzroča nekaj podvajanja. Prav to opažanje pa tudi nakazuje drugo možnost. Na vmesnik za upravljanje lahko gledamo kot na ostale fizične vmesnike. Povežemo ga na most in obravnavamo povsem enakovredno ostalim. Tak pristop je prikazan na sliki 39.

Izvedba na en ali drugi način načeloma ne pomeni funkcionalnih razlik.



Slika 39: Shematski prikaz obravnave upravljavskega prometa skozi psevdovmesnik.

8.5 Obstoječe rešitve

Našteli smo naloge, ki jih opravljajo mostovi, in pogledali, katere elemente most mora vsebovati, poglejmo še, kakšne so praktične izvedbe.

8.5.1 Strojne izvedbe

Ponudniki omrežne opreme za hrbtenična omrežja (angl. *backbone networks*) se odločajo za strojne izvedbe, torej stikala. Njihova glavna prednost je hitrost. Predvsem največji se odločajo za razvoj lastne strojne opreme (Cisco, Nortell, Motorola, Ericsson), ostali pa kupujejo stikala od drugih proizvajalcev (Broadcom, Infineon, Transwitch).

Po redkih dostopnih podatkih[36] o stikalih podjetij Broadcom in Infineon obdelavo okvirja razdelijo v dve fazi:

- **Filtriranje.** Ta faza vključuje vhodno obdelavo okvirja in ugotavljanje, kaj je z okvirjem treba narediti v nadaljevanju. Okvir v tej fazi pridobi različne oznake, od katerih je odvisna njegova kasnejša usoda.
- **Posredovanje.** V tej fazi stikalo okvir spreminja in posreduje na izhode, lahko pa ga tudi zavrže ali posreduje v ločeno vrsto, od koder nadaljuje pot v CPU.

Kot omenjeno, je glavna težava pri oceni strojnih izvedb skopa količina informacij, ki so dostopne o dejanski izvedbi. Proizvajalci načrtovalske dokumentacije razumljivo ne razkrivajo, je pa nekatere stvari mogoče razbrati iz uporabniške dokumentacije in opisa vmesnikov.

8.5.2 Programske izvedbe

Po dostopnih podatkih je Iskratel edini proizvajalec v segmentu dostopa (angl. *access*), ki se v nekaterih primerih odloča za programsko izvedbo. Ponudba programskih rešitev je bogata pri proizvajalcih modemov in mostov za domačo rabo in manjše pisarne (angl. *Small Office, Home Office*, SOHO).

Programska izvedba mosta je tudi že dolgo del Linuxovega jedra. V kombinaciji z orodji za virtualizacijo računalnikov in omrežij (VirtualBox, VMWare) je to odličen pripomoček za učenje in testiranje. Linuxova izvedba mosta je vključena tudi v množico brezžičnih

36 Proizvajalci podrobnosti delovanja skrivajo. Grob opis delovanja je večkrat pogojen s podpisom NDA, natančne specifikacije pa so nedostopne.

usmerjevalnikov za domačo rabo.

Pri programskih izvedbah z lahkoto sledimo poti okvirja in spremembam, ki pri tem nastajajo[37].

Programske rešitve trpijo za nekaj težavami, njihovo reševanje pa je pomemben del načrtovanje in izvedbe:

- **preobremenitev CPU.** Ker z okvirji dela glavni procesor, je seveda temu primerno obremenjen. To pomeni, da moramo veliko pozornosti posvetiti varovanju pred preobremenitvijo.
- **Križanje nadzornih in podatkovnih poti.** Na CPU se izvaja koda, ki nadzoruje delo stikalnega dela in stikala v celoti (angl. *data & control plane mixing*). Ob napakah v programski opremi lahko to pripelje do zelo nezaželenih posledic.
- **Zanesljivost delovanja.** Napake pri delovanju programske opreme so pogosto usodne za most in pomenijo ponovni zagon mosta, običajno pa kar celotne naprave. Zaradi tega pride do izpada, ki lahko traja od nekaj sekund pa do nekaj minut.
- **Hitrost.** Po hitrosti programske rešitve ne morejo dosegati strojnih.
- **Trepetanje.** Poleg premoščanja CPU običajno opravlja tudi druge naloge. Če te naloge niso zelo kratke, potem vplivajo na obdelavo okvirjev, kar se na zunaj kaže kot trepetanje (angl. *jitter*). Ta pojav je zelo nezaželen. Pri podatkovnem prenosu ne povzroča večjih težav, ima pa velik vpliv na IP-telefonijo (presekan govor) in video (največkrat opazno kot „kockanje“ slike).

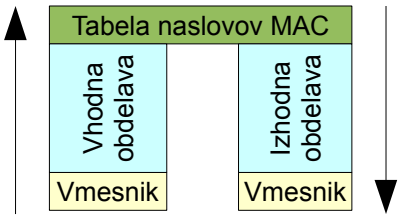
Zgradba programskih mostov odraža naloge (te smo našteli v poglavju *8.3*), ki jih most mora opravljati. Posamezni funkcionalni sklopi imajo tudi pripadajoče sklope v programski opremi. Vrstni red prehajanja skozi funkcionalne sklope v mostu je enak za vse okvirje:

- zaščita CPU pred preobremenitvijo;
- podpora za VLAN-e;
- filtriranje na podlagi različnih lastnosti okvirja;
- preslikava okvirjev za potrebe odkrivanja napak in nadzora;
- zagotavljanje kvalitete storitev –
    - krmiljenje prometa (angl. *policing*) v vhodni smeri in
    - glajenje prometa (angl. *shaping*) v izhodni smeri;
- podpora za skupinsko in razpršeno pošiljanje;
- priprava prometnih statistik;
- tabela naslovov MAC.

Okvir se po prihodu prehaja skozi naštete sklope po predstavljenem vrstnem redu, dokler ne doseže tabele naslovov MAC, ta poskrbi za pošiljanje na pravi izhodni vmesnik ali več vmesnikov, potem pa okvir v obratnem vrstnem redu prehaja iste sklope, dokler končno ni odposlan. Delo mosta lahko torej razdelimo v tri sklope, kot to prikazuje slika 40:

37 Pogoj za to je seveda dostopnost izvorne kode.

- Vhodna (angl. *ingress*) obdelava.
- Tabela naslovov MAC.
- Odhodna (angl. *egress*) obdelava.

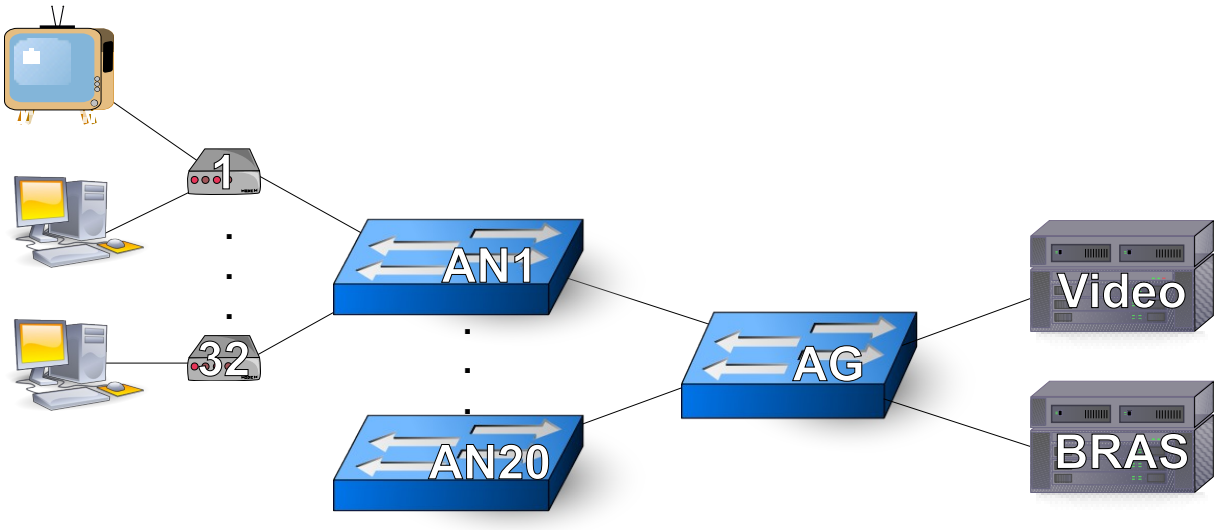


Slika 40: Prehod okvirja skozi most.

8.6 Most v uporabi ponudnika omrežja

Ponudniki omrežij (angl. *network provider*) so pomembni uporabniki mostov. Kljub temu, da slišimo veliko o IP-omrežjih, pa ponudniki omrežje običajno zgradijo iz enega ali več podomrežij na drugem nivoju. Pri tem so mostovi ključnega pomena.

Pričakovanj ponudnikov omrežij ne moremo razumeti, če ne poznamo lastnosti njihovih omrežij. Najbolj nas zanimajo količina in vrste prometa.



Slika 41: Omrežje ponudnika.

Na sliki 41 je grob prikaz dela omrežja tipičnega ponudnika[38]. Na skrajni levi so uporabniki, ki so preko modemov povezani z dostopovnim vozliščem (angl. *Access Node*, AN)[39].

Obstaja več tehnologij digitalnih naročniških vmesnikov (angl. *Digital Subscriber Line*, DSL), vsaka z različnimi hitrostmi prenosa. Nekatere so simetrične, druge asimetrične –

38 Uporabljali bomo podatke, ki se nanašajo na postavitve, ki so enake ali zelo podobne tistim, ki jih imamo v Sloveniji.
39 Izraza AN in DSLAM praktično pomenita isto napravo. Prvega uporabljajo predvsem arhitekti omrežij, drugi pa je domač pri serviserjih in vzdrževalcih omrežij.

hitrosti prenosa od naročnika (angl. *Up-Stream*, US) in proti naročniku (angl. *Down-Stream*, DS) se razlikujejo.

Za potrebe izračuna bomo izbrali tehnologijo VDSL, ki ponuja do 52 Mb/s proti in 16 Mb/s od uporabnika, skupno torej do 68 Mb/s. Pri 32 uporabnikih to pomeni 2176 Mb/s oziroma več kot 2 Gb/s. To je teoretična skrajna količina prometa, ki pa v praksi ni dosegljiva.

Tipičen AN je povezan naprej v omrežje preko 1 Gb/s povezav. Lahko je ena sama, lahko jih je več in podpirajo deljenje prometa (angl. *load sharing*). Privzemimo, da je AN povezan z naslednjim mostom z več takimi linijami, tako da te ne predstavljajo ozkega grla.

Naslednji most je združevalno vozlišče (angl. *Aggregation Node*, AG) in združuje promet več AN (v analiziranih postavitvah jih je lahko do 20). Če smo uspeli promet brez omejitev spraviti čez AN, potem ga je na tem mostu že 43,5 Gb/s. To je velika količina prometa!

Mnogi mostovi na tem nivoju so proti jedru omrežja povezani z 10 Gb/s-povezavami, nekateri pa tudi z manj. Za potrebe analize bomo predpostavili, da je AG proti jedru omrežja povezan s štirimi 1 Gb/s-povezavami, skupno torej 4 Gb/s prenosa.

8.6.1 Poddimenzioniranje omrežja

Na mestu je vprašanje, kakšen smisel ima omrežje, ki ni sposobno naprej prenašati podatkovnega toka uporabnikov.

Omrežje je večino časa malo uporabljano. Televizijo gledamo le nekaj ur na dan, tudi telefonskih pogovorov ne opravljamo brez premora. Ko naložimo stran v brskalnik, jo nekaj časa beremo. Ko omrežje potrebujemo, pričakujemo hiter odziv in veliko prepustnost, večino časa pa ga ne potrebujemo. Verjetnost, da bi omrežje potrebovali vsi uporabniki naenkrat, je majhna[40].

To je podlaga za poddimenzioniranje omrežij. Omrežje mora zagotoviti spodnjo mejo prepustnosti, preostanek pa se uporabi po načelu najboljših možnosti (angl. *best effort*).

Naloga zagotavljanja kvalitete storitev je, da pasovno širino porazdeli med storitve tako, da jo najbolj nujne dobijo zagotovo (IP-telefonija), naslednje v vrsti so srednje prioritetne storitve (IP-televizija), preostanek pa je namenjen podatkovnemu prenosu.

Zakaj naročniki pristajajo na to? Razlog je seveda cena. Pri ponudniku dostopa lahko zakupimo tudi zagotovljeno pasovno širino, a to ima svojo ceno, ki je za večino naročnikov previsoka.

8.6.1.1 Asimetričnost prometa

Ena od osnovnih lastnosti porazdelitve prometa je njegova asimetričnost. V večini primerov veliko več prometa teče v eno kot v drugo stran. Ta lastnost je tako prevladujoča, da je celo dala ime družini tehnologij (ADSL, asimetrični DSL). Domači uporabniki so v veliko večji meri porabniki vsebin, ki se nahajajo v omrežju, kot pa njihovi proizvajalci, in to se odraža tudi v prometu. Naročnik v omrežje pošilja predvsem nadzorni promet (kaj, kdaj, kako), iz omrežja proti njemu teče zahtevani podatkovni tok.

---

40 Majhna, a ne enaka 0. Tako se ob špicah zgodi, da je odzivnost in prepustnost omrežja za posameznega uporabnika manjša kot običajno. Z raznimi testi hitrosti (recimo [56]) lahko izmerimo trenutno hitrost, ki le v najboljšem primeru dosega tisto, ki smo jo ponudniku plačali.

8.6.1.2 Oddaja več prejemnikom

Oddaja istega podatkovnega toka več prejemnikom (angl. *multicast*) je razlog, da sodobna omrežja delujejo tudi takrat, ko bi po prej prikazanih računih moralo priti do zasičenja. Oddaja več prejemnikom omogoča, da promet, ko smo ga le enkrat prejeli, oddamo več naročnikom naenkrat. Na ta način smo vhodni vmesnik obremenili le enkrat.

8.6.2 Zahtevana prepustnost omrežja

Slovenski ponudniki omrežja ponujajo:

- internetno televizijo (2.5 Mb/s),
- IP-telefonijo (zanemarljivo) in
- internetni podatkovni dostop (preostala pasovna širina).

Po grobi oceni torej potrebujemo manj kot 3 Mb/s na povezavo oziroma 96 Mb/s na celoten AN. Na naslednjem nivoju stikal to pomeni 2 Gb/s prometa, a le ob predpostavki, da niti dva uporabnika od 640 ne gledata istega TV-kanala.

Iz tega lahko izračunamo zahtevane prepustnosti mostov na različnih nivojih. AN mora zagotoviti prepustnost vsaj 100 Mb/s oziroma 10.000 okvirjev v sekundi (angl. *frame per second*, FPS). Na naslednjem nivoju je zahtevana prepustnost 2 Gb/s oziroma 200.000 FPS.

Pri izračunu smo predpostavili, da so okvirji veliki 1500 zlogov (pri video prometu je to sprejemljiva predpostavka), vse zaokrožitve pa so navzgor. Da zagotovimo dovolj prostora za poraste (zelo dinamične scene) prometa in nekaj varnostnega prostora, je smiselno, da izračunane vrednosti podvojimo.

Ciljna prepustnost je torej:

- prvi nivo (AN) 200 Mb/s oziroma 20.000 FPS in
- drugi nivo (AG) 4 Gb/s oziroma 400.000 FPS.

8.7 Pomanjkljivosti obstoječih rešitev

Čas je, da se dotaknemo pomanjkljivosti obstoječih rešitev. Te so glavni motiv, da želimo razviti boljšo rešitev.

Najprej bomo na kratko opisali tri mostove, ki jih bomo vključili v primerjavo, sledi pa primerjava lastnosti.

8.7.1 Primerjani mostovi

8.7.1.1 Most operacijskega sistema Linux

Na kratko smo most, ki je dela operacijskega sistema Linux, že omenili v preteklih poglavjih. V primerjavo ga bomo vključili predvsem zato, ker gre za primer zelo odprte arhitekture. Ne le, da je njegova celotna izvorna koda prosto dostopna, pomembnejše je dejstvo, da ta most podpira majhno podmnožico naštetih funkcionalnosti, za vse ostale pa morajo poskrbeti drugi

moduli. Most podpira le poizvedbo v tabeli naslovov MAC in prekinjanje zank z uporabo protokola STP.

Glavna moč tega mosta je v ideji, da lahko infrastrukturo, ki je namenjena podpori omrežnih protokolov višjih nivojev, enakovredno uporabimo tudi za izvedbo posameznih delov mosta. Vse, kar zares potrebujemo, je jasen vmesnik med funkcionalnimi sklopi, potem pa jih lahko razvijamo in uporabljamo neodvisno drugega od drugega. To poenostavi in skrajša razvoj in testiranje, obenem pa dopušča možnost, da bodo uporabniki tako pripravljene komponente uporabili tudi na načine, ki jih razvijalec niti ni predvidel[41].

Zadnja različica mosta operacijskega sistema Linux je vedno na voljo preko spleta. Nahaja se v imeniku `net/bridge` znotraj paketa izvorne kode [41].

8.7.1.2 Visoko integriran most

Drugi most, ki ga bomo vključili v primerjavo, pripada produktu, ki se je že dokazal v postavitvah na terenu. Nastal je pred leti, potem pa doživel več popravkov in dodelav. S stališča arhitekture je popolno nasprotje prej omenjenega mosta. Zasnovan je bil kot monoliten blok, ki mora podpreti celoten nabor zahtevanih funkcionalnosti. Spisek le-teh je skozi čas naraščal, tako da most danes podpira večino funkcionalnosti, ki smo jih našteli. Poleg tega podpira tudi premoščanje okvirjev protokola Ethernet preko omrežnega protokola ATM[42]. To še dodatno poudarja visok nivo integracije.

Slednje pa je tudi šibka točka tega mosta. Zasnovan je bil z mislijo na zelo konkreten problem – in nobenega drugega. Zaradi tega so se kasnejše dodelave izkazale za nerodne. Visoka integracija je bila tudi prepogost izgovor za slabo definirane vmesnike med posameznimi deli.

Avtorji tega mosta so tudi na napačen način razumeli nasvet Michaela A. Jacksona[43]. Tako so se z raznimi optimizaciji ukvarjali že od začetka, a vsak na svoj način, ob branju kode pa se človek le težko znebi vtisa, da si niso vsaj nekateri med njimi vmes tudi premislili.

8.7.1.3 Nizko integriran most

Zadnji most je novejši, pravi preizkus na terenu pa ga še čaka. Pri nastajanju si je ideje izposojal od operacijskega sistema Linux in njegovega mosta, vseeno pa se ni mogel povsem ločiti od svojega visoko integriranega predhodnika.

Nastala kombinacija arhitekturno ni tako odprta kot Linuxov most, še vedno pa je veliko bolj odprta kot visoko integrirani most. Funkcionalno je slednjemu zelo blizu, ne deli pa njegove tesne navezanosti na specifično strojno opremo. Pravzaprav je – tako kot Linuxov most – povsem neodvisen od nje.

8.7.2 Spremembe funkcionalnosti

Dodajanje in spreminjanje funkcionalnosti je glavna težava, za katero trpijo obstoječi mostovi in stikala.

---

41 Taka uporaba ima pogosto negativen predznak, a na tem mestu jo omenjamo predvsem z mislijo na odprte možnosti eksperimentiranja, ki jih tak pristop ponuja.

42 Protokol ATM skrbi za prenos celic pri tehnologijah ADSL in SHDSL.

43 Ne gre za slavnega glasbenika, pač pa za angleškega znanstvenika. Pravila optimizacije je strnil v „nikar“ in „ne še“ (angl. *Rule 1: Don't do it. Rule 2 (for experts only): Don't do it yet.*).

Pri mostovih je težava nekaj manjša, saj je treba spremeniti le izvorno kodo, ki je na voljo. Vendar pa tudi to pomeni veliko testiranja, preden je zaupanje v novo rešitev zadostno, da je ta primerna za namestitev pri strankah. Testiranje je zamudno, poleg tega pa ne zagotavlja pravilnosti delovanja. Zaradi tega so spremembe, še posebej če se potreba po njih pojavi pozno v razvoju novega produkta, drage in zelo nezaželene.

Pri stikalih je težava še večja. Tam spremembe pogosto sploh niso sprejemljive, saj je za njihovo izvedbo treba spremeniti strojno opremo, to pa je drago in v primerih, ko se oprema že nahaja na terenu, tudi neizvedljivo. Po dostopnih podatkih analizirana stikala tudi ne omogočajo pravilne izvedbe nekaterih protokolov [8].

8.7.2.1 Most operacijskega sistema Linux

Posegi v Linuxov most so preprosti, zahtevajo pa spremembo izvorne kode, prevajanje in občasno ponovni zagon[44].

Linuxov most sam po sebi ni nič posebnega s stališča dodajanja funkcionalnosti. Dosti bolj pa je v tem pogledu zanimiv celoten omrežni sklad. Ta namreč omogoča nadvse preprosto povezovanje različnih funkcionalnosti in gradnjo vedno bolj zapletenih postavitev. To univerzalnost smo si med drugim vzeli za zgled in jo zato v poglavju 7.4 podrobneje predstavili.

8.7.2.2 Visoko integriran most

Visoko integriran most je glede spreminjanja in dodajanja funkcionalnosti zelo neroden. Most je sicer programski, težava pa je v tem, da njegova zasnova ni modularna. Ta težava se je še stopnjevala že med razvojem, ko je vsak razvijalec v svojem delu uveljavil svoj pristop, ob tem pa ni bilo dovolj strogega celostnega nadzora.

Danes v ta most še dodajamo funkcionalnosti, vendar pa vsak tak poseg traja veliko dlje kot na novejšem nizko integriranem mostu. Ob tem spremembe zahtevajo tudi veliko testiranja. Dokumentacija mnogim spremembam, ki jih je prinesel čas, ni sledila, pri vzdrževanju prvotni avtorji ne sodelujejo več, to pa vzdrževalce postavlja pred poseben izziv.

8.7.2.3 Nizko integriran most

Dodajanje funkcionalnosti v most je razmeroma lahko. V njegovo načrtovanje in izdelavo so bile vključene mnoge lekcije, pridobljene pri razvoju in uporabi visoko integriranega mosta. Je visoko modularen in enotno oblikovan.

Edini minus dodajanja in spreminjanja funkcionalnosti v ta most je dejstvo, da zahteva spremembe v izvorni kodi osnovnega mosta, to pa za seboj vsakič potegne veliko testiranja. Sprememb ni mogoče testirati brez ponovnega zagona.

8.7.3 Odkrivanje napak

Mostovi in stikala podpirajo velik obseg funkcionalnosti, temu primerno pa je tudi število namestitev. Nekatere od njih so neodvisne, druge ne. Kaj se dogaja s posameznim okvirjem, kam in kakšen bo prišel, je odvisno od globalnih nastavitev mosta in od nastavitev vsaj dveh vmesnikov – vhodnega in izhodnega. Ne le, da morajo biti vse nastavitve pravilne, nastavitve na različnih vmesnikih se morajo ujemati.

44 Z uporabo orodja `ksplice` se je temu mogoče izogniti.

Težava, na katero naletijo začetniki ob nastavljanju mostov, izkušeni uporabniki pa ob dodajanju novih naprednih storitev, je, da je ob nedelovanju težko odkriti razlog. Pri mostovih je težavnost odkrivanja odvisna od avtorjev, ne smemo pa pozabiti, da preprostost odkrivanja napak v nastavitvah ni med prednostnimi nalogami ob načrtovanju[45].

8.7.3.1 Most operacijskega sistema Linux

Linuxov most je pri odkrivanju napak težaven. Glavni izgovor za to je, da gre za odprto kodo, ki jo lahko vsakdo pregleda, posledično pa naj bi bil primerno prečiščen. Pri njegovem delovanju res nismo naleteli na težave, pri dodajanju funkcionalnosti pa redkobesednost glede delovanja ni priročna. Začetnik v primeru napak v nastavitvah tudi težko odkrije, kje in kaj je šlo narobe.

8.7.3.2 Visoko integriran most

Most od začetka ni predvideval pripomočkov za odkrivanje težav. Kasneje so bili na na nekaterih mestih dodani, a nedosledno, predvsem pa tam, kjer so se težave že pojavile. Odkrivanje napak je zato težavno, dodatno pa ga otežujejo že prej omenjene zasnovne in izvedbene težave.

8.7.3.3 Nizko integriran most

Most je poskušal odpraviti napake visoko integriranega mosta. Spremljanje delovanja in poročanje o napakah je bilo vključeno že od začetka, z rastjo števila razvijalcev pa je tudi tu prišlo do odstopanj.

Danes ta most dopušča možnost precej natančnega sledenja dogajanju tako med nastavljanjem kot med delovanjem, ima pa nekaj slepih točk.

Omembe vredno je, da je na možnost odkrivanja napak v tem mostu vplivala tudi ta magistrska naloga.

8.7.4 Podvajanje

V več primerih opazimo, da je potrebno določeno podvajanje funkcionalnosti: sklopa filtrov in zagotavljanja kvalitete storitev delujeta po principu izbira-akcija, vendar kot dve ločeni komponenti pogosto ne delita funkcij za izbiro. Filtri in skrivanje sosednjih vmesnikov imajo veliko skupnega. Filtri so podlaga tudi za napredne funkcije razvrščanja prometa v VLAN-e.

Takemu podvajanju se želimo izogniti. Čeprav so naši motivi predvsem čistoča (med drugim podvajanje implementacije hitro privede v dve različni implementaciji nečesa, kar bi moralo biti enako) in od tod izvirajoča večja preglednost in lažje vzdrževanje.

Ob analizi smo naleteli predvsem na dva razloga za podvajanje:

- Hitrost – pogosto se je razvijalec odločil, da bo določeno funkcionalnost posebej priredil svojim potrebam in jo tako naredil malenkost hitrejšo od bolj splošne izvedbe.
- Pomanjkanje sodelovanja – dva razvijalca sta sočasno ali pa le nevede drug za drugega naletela na isti problem in ga potem oba rešila. Včasih vsak na svoj način, včasih pa na enak način.

45 Mogoče lahko tudi temu pripišemo vedno hujše primere napak pri nastavitvah omrežne opreme [53], [54].

8.7.4.1 Most operacijskega sistema Linux

Linuxov most ponuja zelo majhen del funkcionalnosti sodobnega mosta. Podpira le usmerjevalni del in prekinjanje zank z uporabo protokola STP. Podpora za označevanje VLAN je na voljo v obliki ločenega modula, ki ni del mosta.

Linuxovo stikalo nima težav s podvajanjem, v veliki meri pa gre to tudi na račun zelo omejene funkcionalnosti.

8.7.4.2 Visoko integriran most

Ta most nima veliko podvajanja na nivoju funkcionalnosti, ga je pa veliko na področju podatkovnih struktur in funkcij za delo z njimi. Ob vzdrževalnih posegih je to ena večjih težav.

8.7.4.3 Nizko integriran most

Na tem mostu je kar nekaj podvajanja. Na več mestih je izveden razrez okvirjev (v okviru filtrov, v okviru podpore aplikacijskih protokolov), delo s seznami (filtri, tabela naslovov MAC) ... Nivo podvajanja je trenutno še majhen, a gre za razmeroma mlad most.

8.7.5 Hitrost

Mimo hitrosti preprosto ne moremo. Ne smemo pozabiti, da ne govorimo o osebnih računalnikih, kjer procesor z delovnim taktom 1–3 GHz obdeluje prometni tok enega omrežnega vmesnika, ob tem pa večina uporabnikov niti ne bi opazila nihanj v prepustnosti.

Pri omrežni opremi je treba dostikrat sklepati kompromise. Več funkcionalnosti običajno pomeni večje zahteve po procesorski moči, to pa zahteva nakup močnejšega in tudi dražjega procesorja. Želja po več lahko nasede v preplitvi denarnici.

8.7.5.1 Most operacijskega sistema Linux

Linuxov most je zagotovo najhitrejši med vsemi tremi, a gre to predvsem na račun omejene funkcionalnosti. Primerjava z ostalima dvema tako ne more biti povsem poštena.

Na srečo smo pred leti vseeno imeli možnost testirati visoko integrirani most na enaki strojni opremi in v vsaj delno primerljivih pogojih. V obeh primerih smo uporabljali podporo za VLAN-e. Linuxov most se je takrat odrezal odlično in za več kot dvakratnik prehitel visoko integrirani most.

Vseeno pa ne bi bilo prav, če bi Linuxov most le nekritično hvalili. Pohvalili smo že njegovo odprtost, a ta gre prav na račun hitrosti. Linuxovo jedro je kompromis med univerzalnostjo in hitrostjo. Ob tem, ko je način povezovanja naprav in protokolov zelo priročen, pa je tudi počasnejši, kot bi si želeli[46].

8.7.5.2 Visoko integriran most

Ta most je bil pisan z mislijo na optimizacijo, a na napačen način. Kaže, da je vsak od avtorjev – teh je bilo najbrž preveč – imel svojo idejo o tem, kaj in kako optimizirati. Večkrat je vidno pomanjkanje razmisleka o delovanju mosta kot celote.

46 Razlog za počasnost je skrit v implementaciji funkcije `netif_receive_skb()` v `net/core/dev.c`, ki je odgovorna za posredovanje okvirjev povezanim napravam in protokolom.

Na prej omenjenem primerjalnem testu z mostom operacijskega sistema Linux se je to tudi dobro pokazalo.

V bran temu mostu je zagotovo treba dodati, da je spekter nalog, ki jih opravlja, širši kot na ostalih dveh, saj vključuje tudi navezavo na prenosni nivo ATM.

8.7.5.3 Nizko integriran most

Ta most je hitrejši, kot je bilo pričakovati na podlagi izkušenj z visoko integriranega mosta. Razloge gre delno iskati v orodjih, ki jih ponuja Linuxovo jedro[47], avtorji mosta visoko integriranega mosta pa jih niso implementirali sami.

Seveda je tudi nizko integriran most doživel nekaj optimizacij, nekatere od njih pa prinašajo opazne pospeške. Najbolj pomemben je način sprejema in posredovanja okvirjev. Sprejema jih neposredno od gonilnikov omrežne opreme in jim jih tudi neposredno predaja.

47 Najpomembnejši pri tem so predpomnilniški fondi (angl. *cache pool*), ki močno zmanjšajo potrebo po ponavljajočem zaseganju in sproščanju pomnilnika.

9 Prilagodljivi modularni most

Doslej smo si ogledali zgradbo omrežij, spoznali elemente, ki sestavljajo omrežje in si podrobneje ogledali stikala in mostove. V poglavju 8.7 smo našteli pomanjkljivosti obstoječih rešitev.

Nekatere težave lahko pripišemo napakam, zagrešenim ob izvedbi (podvajanje funkcionalnosti, težko odkrivanje napak), druge pa izvirajo iz arhitekture mosta. Način, kako so na najnižjem nivoju izvedeni deli mosta, preveč dobesedno odraža pogled višjih nivojev[48]. Tako odpade potreba po prevedbi višjih nivojev v nižje, a je zaradi tega most pretirano zapleten.

Predstavili bomo prilagodljivi modularni most, ki odpravlja prej naštete pomanjkljivosti.

- Njegova glavna značilnost je zamenjava visokonivojskega pogleda za nizkonivojskega, s tem pa poenostavitev.
- Zagotovili bomo popolno pouporabljivost vsakega elementa in s tem odpravili podvajanje.
- V osnovno arhitekturo bomo vgradili funkcije za sledenje dogajanju v mostu. Te bodo že brez dodatnih kasnejših nadgradenj omogočale zelo natančno spremljanje dogajanja v mostu in posledično olajšale odkrivanje napak v mostu.
- S splošnimi funkcijami bomo močno zmanjšali potrebo po dodajanju novih funkcij.
- Za primere, ko to ne bi bilo dovolj, pa bo vseeno na voljo razširljiv vmesnik, preko katerega bo nove funkcije mogoče dodati. Ta vmesnik bo obenem uporabljen tudi za vključitev osnovnih funkcij v most.

9.1 Zaželene lastnosti prilagodljivega mosta

V okviru tega dela želimo predstaviti arhitekturo programsko izvedenega prilagodljivega modularnega mosta. Glavno vodilo pri zasnovi je odprtost za prihodnje spremembe.

Obstoječe rešitve podpirajo trenutno zahtevane funkcionalnosti, niso pa pripravljene za vključevanje bodočih funkcionalnosti. Te vstopijo naknadno kot dodatki z več ali manj sreče pri vključevanju v obstoječo arhitekturo.

To sliko želimo obrniti na glavo. Naša zasnova še vedno sledi osnovnemu modelu mosta s slike 40, vendar pa vse nastopa kot del razširitev, ki jih odprta arhitektura podpira.

Z analizo operacij, ki jih nad okvirji izvajajo običajna stikala in mostovi bomo ugotovili, katere akcije moramo podpreti. Prav tako bomo ugotovili tudi, katere kontrolne akcije so potrebne za nadzor prehoda okvirja skozi naš most.

48 Kot primer lahko navedemo storitve (angl. *services*), ki se natančno prilegajo označbam VLAN. Drug tak primer je spreminjanje delov označbe VLAN, ki ga je mogoče pokriti z ločenimi funkcijami za vsakega od treh delov, lahko pa isto opravimo tudi s kombinacijo maske in vrednosti.

9.1.1 Prilagodljivost

Razviti želimo kar se da prilagodljivo arhitekturo mosta. Zaradi potreb vzdrževalcev in uporabnikov omrežij se mostovi še vedno razvijajo. Nekatere od teh funkcionalnosti izdelovalcem mostov posebej natančno predpisujejo, na kakšen način jih je treba vključiti v obstoječe okolje.

Odličen primer je *protokol za obvladovanje napak povezave* (angl. *Connectivity Fault Management*, CFM, [8]). Za pravilno delovanje mora prestrezati okvirje takoj po sprejemu, jih obdelati, vrniti v tok, potem pa iste pakete znova prestreči, tik preden jih s pomočjo tabele naslovov MAC razvrstimo na izhodne vmesnike. Tam jih je treba spet prestreči takoj po razvrstitvi na vmesnike, obdelati, vrniti v tok, znova prestreči tik pred oddajo, zopet obdelati in končno poslati.

9.1.2 Odprtost/modularnost

Most, ki je prilagodljiv, zahteva pa stalne posege v svoje jedro, predstavlja veliko breme za vzdrževalce. Zaradi tega želimo jedro ohraniti majhno in čim bolj stabilno, z odprto arhitekturo pa omogočiti kasnejše dodajanje funkcionalnosti. Tako tudi olajšamo sočasno delo več razvijalcev, saj ne „*hodijo hkrati po isti kodi*“, pač pa vsak lahko razvija svoj del, zanašajoč se na pripravljene vmesnike.

9.1.3 Minimalizem

Most lahko deluje na podlagi velikega števila visoko specializiranih funkcij, lahko pa uporabimo manjše število parametriziranih funkcij.

Kot primer poglejmo nastavljanje prioritete, bitov DEI/CFI in številke znotraj značke VLAN, ki ga lahko podpremo s tremi različnimi funkcijami. Lahko pa to naredimo z eno samo, ki poleg nove vrednosti vsebuje še masko, s katero ohranimo ali izbrišemo del stare vrednosti značke VLAN.

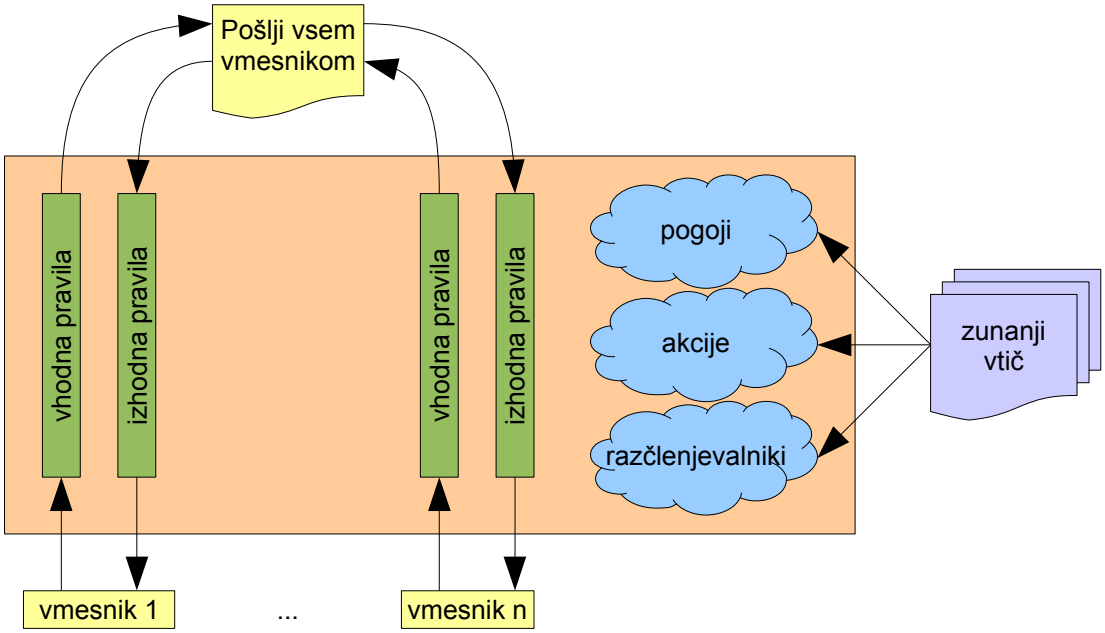
Deli mosta lahko uporabljajo sistemske knjižnice in funkcije v njih, lahko pa iz različnih razlogov definirajo svoje funkcije. Te so včasih boljše, ker so specializirane za točno določeno nalogo, vendar pa pomenijo podvojeno funkcionalnost.

Glavna težava podvajanje je, da nosi s sabo skrite stroške in nevarnosti. Vzdrževanje več funkcij, ki delajo praktično isto, pomeni nepotrebne stroške. Nevarno pa je, kadar ob takih posegih ne uspemo popraviti vseh funkcij, in rezultat je neskladno delovanje delov sistema.

9.2 Arhitektura

Arhitektura univerzalnega mosta je prikazana na sliki 42, sestavljata pa ga:

- ogrodje, ki omogoča dodajanje pogojev in akcij. Ti so uporabljeni za definicijo pravil za obdelavo prometa.
- Jedro, ki izvaja vhodna in izhodna pravila.
- Razpošiljanje na vse vmesnike, ki so člani mosta.



Slika 42: Arhitektura prilagodljivega modularnega mosta.

Tabelo naslovov MAC smo zamenjali s pošiljanjem na vse vmesnike. To pomeni, da se bo naš most privzeto obnašal kot spojnik. Taka izvedba zagotavlja pravilno delovanje na najbolj preprost mogoč način.

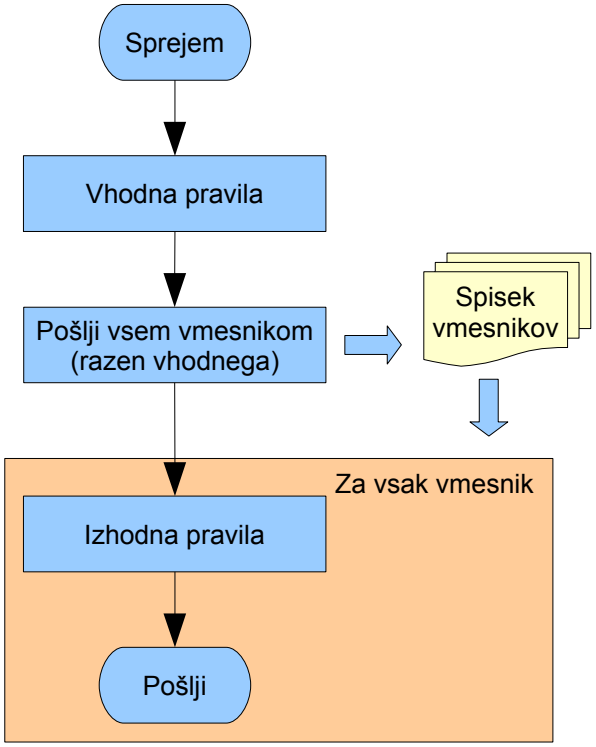
Funkcionalnost tabele naslovov MAC bomo kasneje vseeno vpeljali, a kot akcijo, ki jo lahko izvedemo nad poljubno izbranim prometom.

9.2.1 Srce mosta

Srce univerzalnega mosta je zelo majhno in podpira le:

- programski vmesnik, preko katerega lahko drugi moduli prijavijo in odjavijo pogoje, akcije in razčlenjevalnike ukazov,
- upravljanje spiskov vhodnih in izhodnih akcij za vsak vmesnik posebej in
- majhno jedro, ki vsak prejeti okvir vodi skozi spisek vhodnih akcij, ga razmnoži na vse preostale vmesnike, ga vodi preko spiska izhodnih akcij in ga na koncu odda.

Glavna naloga osrednjega dela univerzalnega mosta je izvedba programskega vmesnika, preko katerega drugi moduli lahko razširijo njegovo zelo osnovno funkcionalnost. Most je tako razširljiv in prilagodljiv. Pomembno je, da so spremembe mogoče med delovanjem in ne zahtevajo ponovnega zagona mosta. Naloga jedra je tudi, da omogoči pregled nad dogajanjem v mostu.



Slika 43: Obdelava okvirjev v prilagodljive modularnem mostu.

Jedro mosta je prikazano na sliki 43, algoritem izvajanja pa je sledeč (slika 44):

```
bridge(ingress_port, received_frame)
{
     // Perform input processing.
     ingress_frame = ingress_processing(ingress_port,
               received_frame);
     if (ingress_frame not null) {
          // Perform MAC table lookup and find list of egress
          // ports.
          egress_ports = get_all_bridge_ports_except_ingress(
                    ingress_port);
          for (each egress_port in egress_ports) {
               // Perform output processing.
               egress_frame = egress_processing(
                         egress_port, ingress_frame);
               if (egress_frame not null) {
                    // Send frame.
                    send_frame(egress_port, egress_frame);
               }
          }
     }
}
```

Slika 44: Algoritem jedra mosta.

9.2.2 Odsotnost tabele naslovov MAC

Marsikoga ob pogledu na predstavljeno arhitekturo najbrž zbode v oči odsotnost tabele naslovov MAC. Doslej smo jo povsod predstavljali kot enega ključnih elementov mosta, v predstavljeni arhitekturi pa je tako pomemben element povsem odsoten. Nadomešča jo pošiljanje vseh okvirjev vsem povezanim napravam, a je tudi to vključeno le zaradi priročnosti, ne pa kot nujno potreben del.

Naš namen je nadomestiti različne korake v obdelavi okvirjev s pogoji in akcijami, ki jih lahko poljubno sestavljamo. Tudi koraka, za katera običajno skrbi tabela – učenje naslovov in razpošiljanje na podlagi ciljnega naslova –, bomo uvedli kot akciji, ki osnovno delovanje mosta razširjata, ne pa kot njegovo osnovo.

Doslej smo že spoznali, da obstajata vsaj dve različici tabele naslovov MAC: taka, ki se zaveda označitev VLAN, in taka, ki se jih ne. S tem dopuščamo možnost, da lahko celo znotraj enega mosta, vsekakor pa na več mostovih, preizkušamo obe različici.

Odstranitev tako ključnega elementa priča tudi o odločenosti, da možnosti, ki jih ponujajo vhodno-izhodna pravila, izkoristimo do konca.

9.2.3 Vtiči

Glavna moč prilagodljivega modularnega mosta je v vtičih (angl. *plug-in*), ki jih lahko uporabnik dodaja in odvzema med delovanjem. Ti poljubno razširijo osnovno funkcionalnost mosta, obenem pa puščajo jedro zelo majhno in posledično lažje preverljivo.

Preko vtičev lahko mostu dodajamo nove vrste akcij in nove pogoje. Te lahko potem uporabljamo za vhodno in izhodno obdelavo okvirjev.

Posamezni vtič lahko mostu preskrbi le akcije, le pogoje, ali pa oboje. Za prijavo in odjavo akcij in pogojev vtič uporabi vmesnik, ki ga posebej za to ponuja most.

9.2.4 Operacijski sistem

Izvedba mosta ni vezana na določen operacijski sistem. Potrebujemo sicer nekatere elemente, ki pa so v današnjih operacijskih sistemih stalnica, v primeru odsotnosti pa jih pač moramo preskrbeti sami.

9.2.4.1 Dinamične knjižnice

Dinamične knjižnice niso nujen pogoj za izvedbo prilagodljivega modularnega mosta, so pa pri tem v veliko pomoč. Le tako je mogoče med delovanjem spreminjati funkcionalnost.

Na operacijskem sistemu Linux so že dolgo na voljo – v jedru in za uporabniške programe. Podobno je v družini operacijskih sistemov Windows. Dinamične module podpira tudi operacijski sistem VxWorks [48].

Izvedba prilagodljivega mosta je mogoča tudi v primeru, da operacijski sistem ne podpira dinamičnih knjižnic, le da je za spremembo funkcionalnosti potrebno ponovno prevajanje in običajno tudi ponovni zagon omrežnega dela operacijskega sistema (to ponavadi pomeni kar ponovni zagon celotnega sistema). Si pa taka izvedba težje zasluži vzdevek *modularna*, razen če imamo v mislih modularnost na nivoju programske kode.

9.2.4.2 Delo s pomnilnikom

Operacijski sistem lahko mostu močno pomaga tudi pri upravljanju pomnilnika. Most stalno zasega in sprošča kose pomnilnika, največji del teh operacij pa se nanaša na pomnilnik, v katerem hranimo okvirje. Pri njih je vnaprej znana največja možna velikost, to lastnost pa lahko s pridom izkoristimo. Namesto stalnega zaseganja in sproščanja lahko operacijski sistem ponuja orodja, ki sproščene kose začasno hranijo za primer, da bi jih kmalu zopet potrebovali. Na ta način dvakrat prihranimo čas.

Kjer te podpore ni, imamo na izbiro, da delamo brez nje, lahko pa jo dodamo tudi sami. To terja čas in prinaša nove možnosti za napake. Razvoj tovrstnih pripomočkov ni bistvo te naloge.

9.2.4.3 Sinhronizacija in zaklepanje

Čeprav včasih pod različnimi imeni, pa različni operacijski sistemi vseeno vključujejo osnovne elemente, potrebne za sinhronizacijo in zaklepanje vzporednih izvajalnih niti. Če kakšen operacijski sistem teh elementov ne premore, potem nam ne preostane drugega, kot da jih dodamo sami. Pri tem je potrebno dobro poznavanje operacijskega sistema, procesorja in strojnih ukazov zanj.

Tudi pripomočki za sinhronizacijo in zaklepanje niso predmet te naloge.

9.2.4.4 Navezava na ostale naprave

Operacijski sistem je tisti, ki običajno povezuje omrežne naprave z njihovimi gonilniki na eni strani in navidezne naprave in protokole na drugi strani. Most spada med navidezne naprave.

Operacijski sistemi za povezovanje naprav uporabljajo različne pristope, odvisno od ocene avtorjev, kateri nalogi dati prednost. Linux je dal prednost odprtosti in prilagodljivosti, zato je način povezovanja naprav, navideznih naprav in protokolov zelo fleksibilen – a tudi ne najbolj učinkovit. To so avtorji omrežnega dela jedra spoznali že pred leti in posebej za naprave, kot je most, dodali posebne nastavke[49]. Te uporablja tudi most operacijskega sistema Linux.

Operacijski sistem VxWorks nima te optimizirane možnosti, zato je tam pogosta odločitev, da gonilnik omrežne opreme okvirje predaja mostu neposredno. To seveda oteži integracijo, a gre za zavesten kompromis, ki v ospredje predstavlja zmogljivost.

9.2.4.5 Jedro ali uporabniški prostor

Na nekaterih operacijskih sistemih[50] je pomembno vprašanje, kam sodi most – v jedro (angl. *kernel*) ali uporabniški prostor (angl. *user space*). Če želimo doseči spodobno hitrost, potem vprašanja ni: most mora v jedro. A pomembno je, da se zavedamo, da je hitrost pravzaprav edini razlog, da mora bit most prav tam.

Na operacijskem sistemu Linux brez večjih težav lahko izvedemo most tudi kot aplikacijo v uporabniškem prostoru. To lahko ponuja velike prednosti predvsem pri odkrivanju in odpravljanju napak. V uporabniškem prostoru imamo na voljo zelo priročen razhroščevalnik (angl. *debugger*), orodja za delo v jedru pa so mnogo bolj okorna in dostikrat ogrozijo

49 Pred časom je imelo jedro operacijskega sistema Linux poseben nastavek, ki ga je uporabljal most. Kasneje so se pojavili tudi drugi moduli, ki so imeli enako potrebo po prestrezanju vsega prometa. V zadnjih različicah jedra [41] je zato na voljo univerzalen način povezave omrežnih naprav – sprejemni prestrezniki.

50 Nekateri vgradni operacijski sistemi zaradi hitrosti nimajo ločevanja med jedrnim in uporabniškim prostorom. Tak je lahko VxWorks (odvisno od nastavitev).

stabilnost in obnašanje celotnega operacijskega sistema.

Operacijski sistem VxWorks lahko v celoti teče v enem naslovnem prostoru – v prostoru jedra – vključno z uporabniškimi programi. Na ta način prihranimo dragoceni čas, ki sicer gre na račun prehajanja v uporabniški prostor in nazaj. Cena za to so manjša stabilnost v primeru težav v posameznih procesih, presluhi med procesi in nasploh težave pri odkrivanju napak.

Da ta razmislek ni povsem odveč, konec koncev nakazuje tudi operacijski sistem MINIX [47]. Sestavlja ga zelo majhno jedro, gonilniki naprav in omrežnih protokolov pa že sodijo v uporabniški prostor. Tak sistem je res počasnejši, je pa zelo varen in robusten.

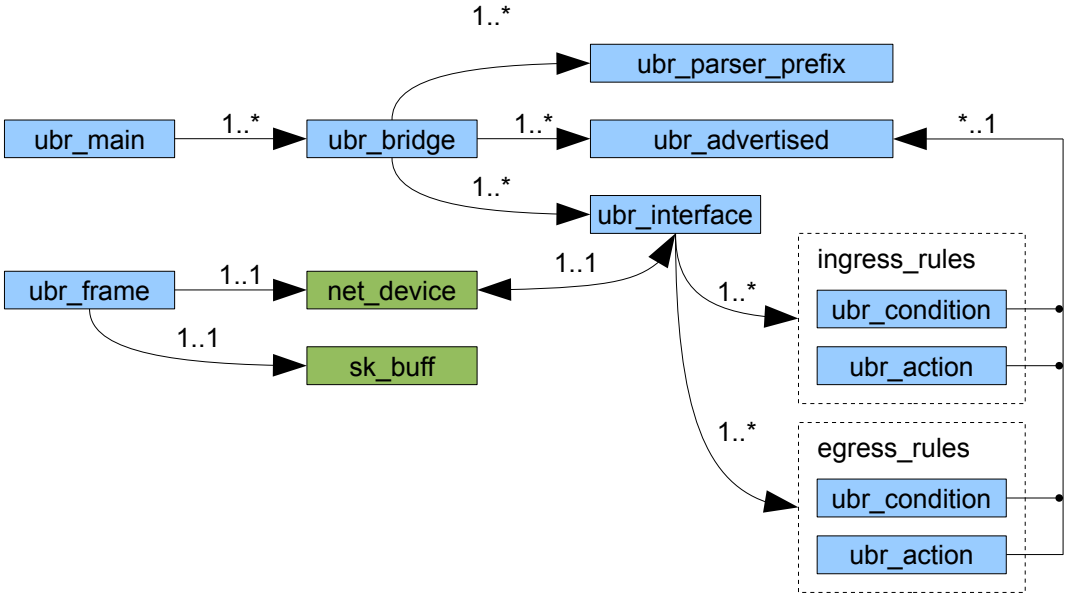
9.3 Prilagodljivi modularni most na operacijskem sistemu Linux

Predstavili bomo podatkovne strukture in programski vmesnik za delo z mostom. Oboje bomo prilagodili operacijskemu sistemu Linux. To ne pomeni, da enakovredne funkcionalnosti ne moremo pripraviti tudi na drugem operacijskem sistemu. Z izbiro konkretnega operacijskega sistema je lahko obravnava bolj oprijemljiva, ker ne govorimo več o povsem abstraktnih pojmih.

Za lažje razumevanje bomo vsem strukturam in funkcijam prilagodljivega modularnega mosta dali predpono `ubr_`. Tako se bomo izognili zmedi, do katere bi lahko prišlo zaradi prekrivanja imen z mostom, ki je že del operacijskega sistema Linux. Ta ima predpono `br_`.

9.3.1 Podatkovne strukture

Na sliki 45 vidimo podatkovne strukture prilagodljivega modularnega mosta in povezave med njimi. Zeleno obarvane strukture so tiste, ki jih ponuja jedro operacijskega sistema Linux, pa tudi na drugih operacijskih sistemih obstajajo njim enakovredne podatkovne strukture.



Slika 45: Podatkovne strukture prilagodljivega modularnega mosta.

Z zeleno barvo sta označeni strukturi, ki ju ponuja jedro operacijskega sistema Linux, mi pa ju bomo le uporabili. V opis ju uvrščamo zaradi pomembne vloge, ki jo igrata.

9.3.1.1 Okvir

Hrambi okvirjev je namenjena struktura `sk_buff`. Ta hrani podatke o kosu pomnilnika, v katerem se okvir nahaja, dolžini okvirja, dodatne podatke o vsebini, vhodnem ali izhodnem vmesniku okvirja itd. Vseh podatkov je preveč za podrobno obravnavo, poleg tega pa za most niso vsi pomembni.

Struktura `sk_buff` je del operacijskega sistema Linux.

9.3.1.2 Omrežne naprave

Omrežne naprave, prave (fizične) in navidezne, hranijo svoje podatke v strukturi `net_device`. Tam so shranjene razne informacije o napravi:

- funkcije, ki jih lahko izvaja naprava,
- statistika naprave,
- pripadnost naprave mostu,
- drugi podatki.

Struktura `net_device` je del operacijskega sistema Linux.

9.3.1.3 Opis stanja obdelave okvirja

Med obdelavo okvirja, premikanjem med akcijami ter pogoji vhodne in izhodne obdelovalne vrste, se moramo nekako zavedati, kje se ta hip nahajamo, kaj obdelujemo... Temu je namenjena struktura `ubr_frame` (slika 46), ki vsebuje naslednje podatke:

- okvir;
- vhodna naprava (v času vhodne obdelave ima to polje vrednost 0, kar omogoča ločevanje med vhodno in izhodno obdelavo);
- številčno oznako trenutnega položaja znotraj vhodne ali izhodne obdelovalne vrste;
- zastavico, ki sporoča, ali je trenutno vključeno sledenje obdelavi.

```
struct ubr_frame {
     struct sk_buff *skb;
     struct net_device *input_dev;
     int index;
     int debug;
};
```

Slika 46: Stanje obdelovanega okvirja.

Med obdelavo lahko pridemo do situacije, ko se odločimo, da okvirja ne bomo obdelovali naprej. Tako odločitev sprejme posamezna akcijska funkcija, jedru mosta pa to sporoči tako, da sprosti okvir in ustrezno polje postavi na 0 (polje `skb` v strukturi `ubr_frame`).

9.3.1.4 Pogoji in akcije

Pogoje in akcije ločimo na dve vrsti:

- **objavljen** je pogoj ali akcija, ki ga vtič podpira in je na voljo za uporabo v vhodni in izhodni obdelavi okvirjev.
- **Uporabljen** je objavljen pogoj ali akcija, uporabljena v vhodni ali izhodni obdelavi okvirja.

O objavljenih moramo poznati naslednje podatke (slika 47):

- modul (vtič), ki prinaša pogoj ali akcijo;
- ime in opis;
- števec uporabe;
- tip;
- funkcija, ki opravlja preverjanje pogoja oziroma izvaja akcijo;
- funkcija, ki iz besedilnega opisa pripravi dejanski opis pogoja ali akcije, in ga vrne v obliki, ki jo lahko uvrstimo v seznam za vhodno ali izhodno obdelavo;
- funkcija, ki izpiše zasebne podatke pogoja ali akcije;
- funkcija, ki sprosti uporabljen pogoj ali akcijo po tem, ko sta bila že odstranjena iz seznama za vhodno ali izhodno obdelavo, in
- zasebni podatki poljubne dolžine (njihova interpretacija je prepuščena posameznemu vtiču).

```
struct ubr_advertised {
     struct hlist_node hlist;
     struct module *module;
     char *name;
     char *description;
     unsigned int use_count;
     enum ubr_entry_type type;
     int (*func)(
               struct ubr_frame *frame,
               struct ubr_entry *entry);
     int (*alloc)(
               struct ubr_advertised *ac,
               char *string,
               struct ubr_interface *ifc,
               struct ubr_entry **new_entry);
     int (*print)(
               const struct ubr_entry *entry,
               char *buffer,
               int buffer_size);
     int (*free)(struct ubr_entry *entry);
     unsigned char private_data[0];
};
```

Slika 47: Objavljeni pogoj.

Števec uporabe je zelo pomemben. Dejstvo, da je pogoj ali akcija objavljena, ne pomeni, da je tudi uporabljena. Če ni uporabljena, potem lahko njen oglas umaknemo. Tudi podatki o modulu so informativne narave, pomagajo pa pri odpravljanju napak.

Uporabljeni pogoji in akcije potrebuje druge podatke (slika 48):

- številčno oznako, s pomočjo katere lahko pogoje in akcije razvrstimo v urejeno zaporedje;
- objavljen pogoj ali akcija, ki je „starš“ tega pogoja ali akcije;
- zastavico, ki označuje, ali gre za normalno ali negirano pravilo (pri akcijah to polje nima pomena); in
- zasebne podatke poljubne dolžine, ki so potrebni za interpretacijo pogoja[51] ali akcije[52].

```
struct ubr_entry {
     struct hlist_node hlist;
     int order;
     struct ubr_advertised *advertised;
     int negate;
     unsigned char private_data[0];
};
```

Slika 48: Uporabljeni pogoj.

Obe strukturi vsebujeta tudi polje `hlist`. Gre za eno od orodij, ki ga ponuja jedro operacijskega sistema Linux, omogoča pa gradnjo enojno povezanih seznamov. Ker smo ti polji dodali v svoji strukturi, bomo lahko uporabljali že pripravljene funkcije za dodajanje, brisanje in sprehajanje po seznamu.

Tudi pogoji lahko spremenijo okvir. Med pogoji in akcijami v resnici obstaja le ena razlika: pri akcijah nas vrnjena vrednost ne zanima, pri pogojih pa nas.

9.3.1.5 Seznam pogojev in akcij

Za vsak vmesnik vzdržujemo dva seznama:

- seznam vhodnih pogojev in akcij in
- seznam izhodnih pogojev in akcij.

Prvega uporabimo ob sprejemu okvirja, ko pridemo do njegovega konca, pa okvir razpošljemo vsem ostalim vmesnikom. Drugega uporabimo pred oddajo okvirja.

V obeh primerih uporabljamo seznam, urejen po naraščajočih vrednostih polja `order`, ki je del opisa posameznega uporabljenega pogoja in akcije.

V istem seznamu hranimo pogoje in akcije, ločimo pa jih preko pripadajočega oglasa.

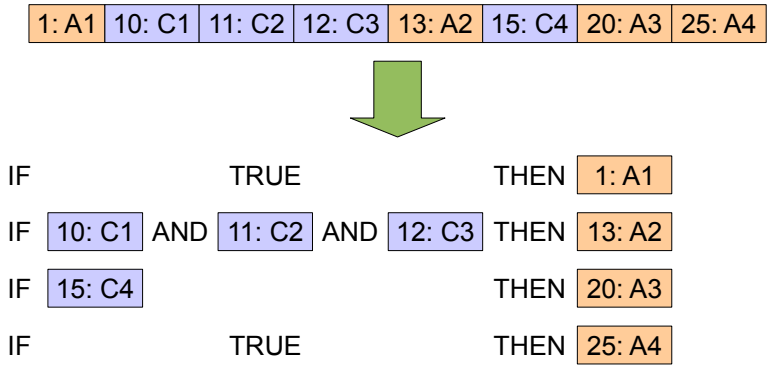
Na sliki 49 vidimo v zgornjem delu primer seznama akcij in pogojev. Akcije so označene z rdečo barvo in črko A, pogoji pa z modro in črko C. Številske predpone predstavljajo vrednost

51 Denimo, da preverjamo ciljni naslov MAC v okvirju. Nekje moramo hraniti podatek o tem, kakšen naslov pričakujemo. Tega hranimo v `private_data`, ki mora v ta namen obsegati vsaj 6 zlogov.

52 Za primer lahko damo akcijo, ki dodaja označbe VLAN. Potrebujemo vsaj en podatek – številko VLAN-a, ki jo bomo zapisali v dodano značko VLAN.

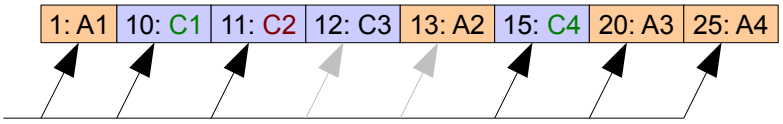
polja `order` – kot vidimo, so naraščajoče.

Seznam je v spodnjem delu razrezan in razložen za lažje razumevanje. Ob obdelavi takega seznama izvajamo pogoje, dokler ne pridemo do akcije. Če tako dosežemo akcijo – uspešni torej morajo biti vsi pogoji –, potem izvedemo vse akcije do prvega naslednjega pogoja.



Slika 49: Obdelava seznama pogojev in akcij.

Ko pridemo do pogoja, znova preverjamo uspešnost vseh pogojev. Če kateri ni uspešen, potem preostale pogoje preskočimo, prav tako pa tudi vse akcije, ki sledijo – do naslednjega pogoja. Tako nadaljujemo, dokler ne dosežemo konca seznama ali pa točke, ko katera od akcij zahteva odmetavanje okvirja (polje `skb` v `ubr_frame` strukturi postavi na vrednost `0`).



Slika 50: Primer obdelave seznama pogojev in akcij.

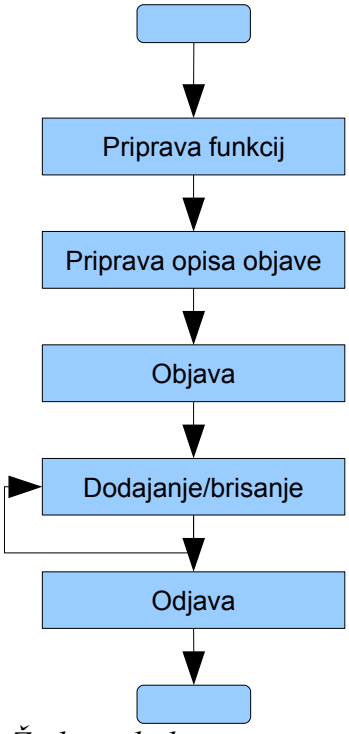
Na sliki 50 vidimo primer obdelave seznama:

- Pred akcijo A1 ni nobenega pogoja, zato se ta akcija vedno izvede.
- Pogoj C1 je bil uspešen, pogoj C2 pa ne. Zaradi tega preskočimo pogoj C3 in akcijo A2.
- Izvajanje nadaljujemo pri pogoju C4, ki je uspešen, zato izvedemo akciji A3 in A4.

9.3.1.5.1 Življenjski krog pogojev in akcij

Za lažje razumevanje se za trenutek posvetimo življenjskemu krogu pogojev in akcij. Doslej smo že videli, kako poteka priprava funkcij, kako te sestavimo v opis objave in kako dosežemo objavo. Ta postopek je prikazan tudi na sliki 51.

Ko je pogoj ali akcija enkrat objavljena, je na voljo za uporabo v vhodnih in izhodnih seznamih za obdelavo okvirjev. To pomeni, da lahko na podlagi objave kdorkoli zahteva uporabo pogoja ali akcije na določenem mestu v vhodnem ali izhodnem spisku enega ali več vmesnikov.



Slika 51: Življenjski krog pogojev in akcij.

9.3.1.6 Predpone razčlenjevalnika

Kasneje bomo predstavili ukazno vrstico, ki jo podpira prilagodljivi modularni most, že sedaj pa bomo opisali podatkovno strukturo, v kateri hranimo predpone, ki omogočajo delo z ukazno vrstico. Za vsako od predpon potrebujemo tudi funkcijo, ki opravi razčlenjevanje preostalih parametrov ukazne vrstice (slika 52).

```
struct ubr_parser {
     struct hlist_node hlist;
     struct module *module;
     char *prefix;
     char *description;
     int (*parser_func)(struct ubr_parser *parser, char
     *parameters,
               char *result, int result_len);
     unsigned char private_data[0];
};
```

Slika 52: Podatkovna struktura na razčlenjevalnika ukazne vrstice.

9.3.1.7 Prijave na obvestila

Nekateri pogoji in akcije za pravilno delovanje potrebujejo informacije o tem, da je bil narejen nov most ali da bo obstoječi most izbrisan. Še več jih potrebuje informacije o vmesnikih, ki so bili dodani ali bodo odstranjeni z mosta. Kdor take informacije potrebuje, jih tudi lahko dobi, mora pa svoj interes oznaniti s prijavo na tovrstna sporočila. Podatkovna struktura, ki pri tem sodeluje, je prikazana na sliki 53.

```
struct ubr_notify {
     struct hlist_node hlist;
     struct module *module;
     const char *func;
     void (*post_bridge_add)(struct ubr_bridge *ubr);
     int  (*pre_bridge_del)(struct ubr_bridge *ubr);
     void (*post_interface_add)(struct ubr_interface *ifc);
     int  (*pre_interface_del)(struct ubr_interface *ifc);
};
```

Slika 53: Podatkovna struktura prijave na obvestila.

9.3.1.8 Na most priključene naprave

Za vsako napravo, ki je priključena na most, moramo hraniti seznam vhodnih in izhodnih akcij. Seveda si zapomnimo tudi napravo samo, pa most, ki mu naprava pripada (dopuščamo več mostov), za konec pa še največje število korakov, ki smo jih pripravljeni opraviti pri obdelavi v vhodnih in izhodnih seznamih (slika 54).

```
struct ubr_interface {
     struct hlist_node hlist;
     struct ubr_bridge *ubr;
     struct net_device *dev;
     struct hlist_head ingress_rules;
     struct hlist_head egress_rules;
     unsigned int max_steps;
};
```

Slika 54: Podatkovna struktura na most priključene naprave.

9.3.1.9 Most

Potrebujemo podatkovno strukturo, v kateri bomo hranili vse podatke, ki se nanašajo na most:

- ključavnico, s pomočjo katere varujemo most pred sočasnimi spremembami,
- omrežno napravo, ki v operacijskem sistemu predstavlja most,
- števec in seznam naprav, ki so povezane nanj, in
- funkcijo, ki opravlja obdelavo prejetih okvirjev.

To strukturo bomo poimenovali `ubr_bridge` (slika 55).

```
struct ubr_bridge {
     struct hlist_node hlist;
     spinlock_t lock;
     struct net_device *dev;
     int if_count;
     struct hlist_head if_list;
     rx_handler_func_t *rx_handler;
};
```

Slika 55: Podatkovna struktura prilagodljivega modularnega mosta.

9.3.1.10 Splošni podatki

Zadnja struktura, ki jo še potrebujemo, je `ubr_main` (slika 56). Namenjena je hrambi splošnih podatkov, ki se nanašajo na vse mostove. Zaenkrat bomo v njej hranili:

```
struct ubr_main {
     struct hlist_head bridges;
     struct hlist_head advertised;
     struct hlist_head parser_prefixes;
};
```

Slika 56: Splošni podatki mosta.

- spisek primerkov mostov[53],
- seznam objavljenih pogojev in akcij in
- seznam predpon, ki jih razume razčlenjevalnik.

9.3.2 Povezava mosta in omrežnih naprav

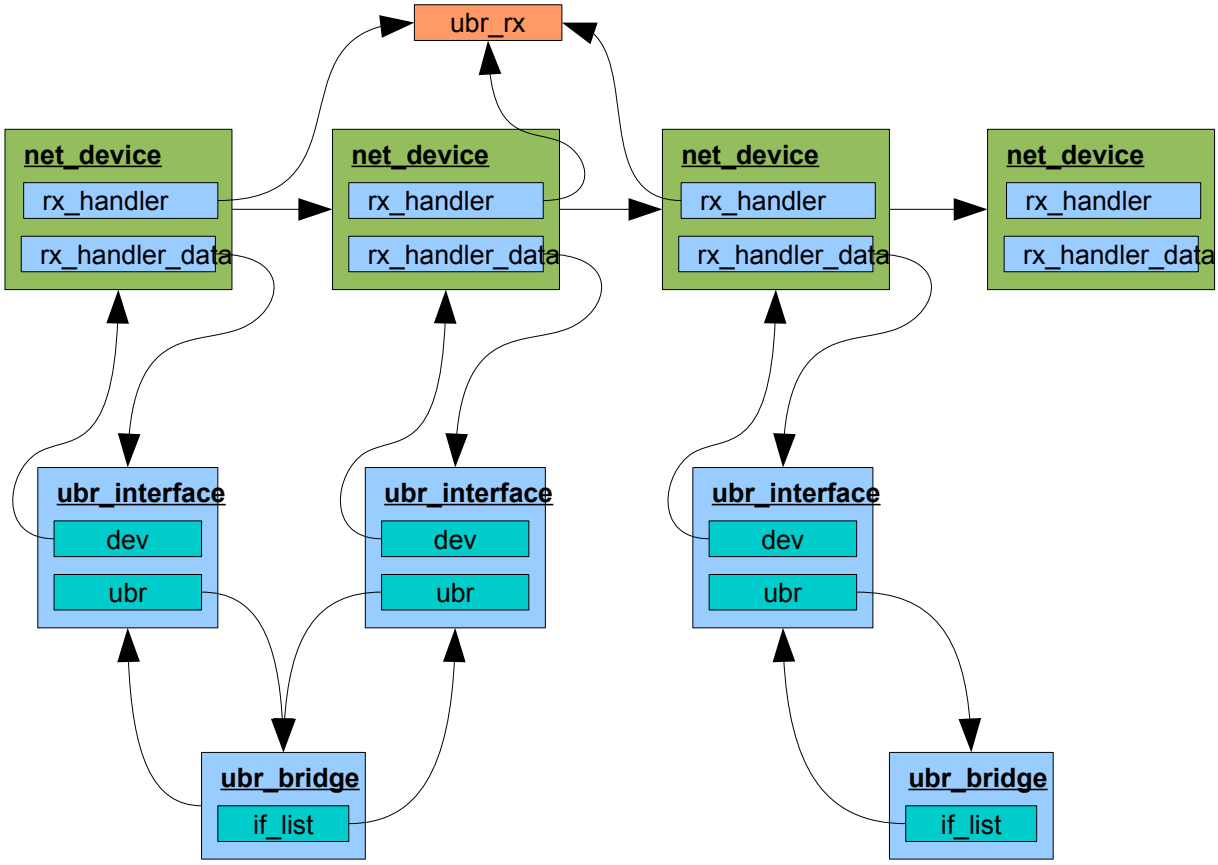
Za povezavo mosta in naprav bomo uporabili enega od mehanizmov jedra operacijskega sistema Linux, ki so za to na voljo. Za vsako od naprav, ki so povezane z mostom, bomo registrirali sprejemni prestreznik `rx_handler`. Na ta način bomo sprejeli vse okvirje, ki jih taka naprava sprejme.

Vsaka omrežna naprava je opisana s strukturo `net_device`, član te strukture pa je poleg prej omenjene `rx_handler` tudi polje `rx_handler_data`. Vanjo bomo shranili kazalec na `ubr_interface` strukturo, ki pripada omenjeni mrežni napravi. Preko nje bomo prepoznali, katere omrežne naprave pripadajo mostu, pa tudi kateremu mostu pripadajo.

Primer povezave treh od štirih omrežnih naprav v dva mosta je prikazan na sliki 57:

- Prvi dve mrežni napravi sta vključeni v prvi most, tretja naprava pa v drugi most. Oba mosta imata tudi vsak svoj spisek omrežnih naprav, ki so vanju vključene. Na ta način je vzpostavljena povezava od naprave do mosta, v katerega je naprava vključena, in od mosta do naprav, ki mu pripadajo.
- Na sliki je prikazan sprejemni prestreznik v obliki funkcije `ubr_rx()`, ki sprejema ves promet v most povezanih naprav.
- V most povezano omrežno napravo prepoznamo po tem, da njeno polje `rx_handler` kaže na naš sprejemni prestreznik (`ubr_rx`). V tem primeru polje `rx_handler_data` kaže na napravi pripadajočo strukturo `ubr_interface`.
- Preko strukture `ubr_interface` lahko vedno pridemo do mosta, ki mu naprava pripada, preko polja `ubr`. Do omrežne naprave pridemo preko polja `dev`.
- V mostu lahko vse povezane naprave vedno najdemo v spisku `if_list`.
- Četrta naprava ni povezana na naš most. Vseeno je mogoče, da je povezana kam drugam. Na istem sistemu je na primer lahko sočasno v uporabi standardni Linuxov

53 Že prej smo omenili, da na primer pri obravnavi značk VLAN obstaja več pristopov. Lahko jih popolnoma ignoriramo, lahko jih podpremo v tabeli naslovov MAC, lahko pa tudi za vsak VLAN naredimo drug primerek mosta.



Slika 57: Povezava omrežnih naprav in mosta.

most, prav tako pa lahko sočasno na istem sistemu združujemo omrežne vmesnike v povezane omrežne vmesnike (angl. *bonding*). A nobena od teh uporab ne bo nastavila enakega sprejemnega prestreznika kot mi – tako lahko vedno prepoznamo v naš most povezane naprave.

9.3.3 Programski vmesnik, namenjen vtičem

9.3.3.1 Pogoji, akcije, razčlenjevalniki

Programski vmesnik za prijavo in odjavo pogojev in akcij je preprost (slika 58). Vtič pripravi opis akcije ali pogoja tako, da izpolni pripadajočo podatkovno strukturo `ubr_advertised`, potem pa pokliče ustrezno funkcijo. V procesu odjave vtič odjavi akcijo ali pogoj uporabljajoč isto podatkovno strukturo.

```
/* Conditions and actions. */
int ubr_advertised_register(struct ubr_advertised *a);
int ubr_advertised_unregister(struct ubr_advertised *a);

/* Command parsing. */
int ubr_parser_register(struct ubr_parser *p);
int ubr_parser_unregister(struct ubr_parser *p);
```

Slika 58: Programski vmesnik za prijavo in odjavo pogojev in akcij.

Vtič lahko prijavi tudi razčlenjevalnik za nov ukaz. Na ta način se vtič vključi v obdelavo ukazov.

9.3.3.2 Primer prijave in odjave pogoja

Poglejmo sedaj primer uporabe doslej predstavljenega programskega vmesnika. Pripravili bomo pogoj, ki preveri, če je paket poslan na ciljni naslov za razpršeno pošiljanje (angl. *broadcast*).

```
static int is_target_mac_broadcast_check(
          struct ubr_frame *frame, struct ubr_entry *entry)
{
     if ( frame->skb->data[0] & 0x01 == 0x01 )
          return 1;
     else
          return 0;
}
```

Slika 59: Preverjanje pogoja.

To prepoznamo tako, da preverimo najmanj pomemben bit prvega zloga ciljnega naslova MAC. Ta se nahaja povsem na začetku okvirja Ethernet. V času obdelave imamo dostop do okvirja preko polja `skb` v strukturi `ubr_frame`. Na začetek okvirja kaže kazalec `data`. Funkcijo `is_target_mac_broadcast_check()`, ki bo dejansko preverjala okvirje, vidimo na sliki 59.

```
static int is_target_mac_broadcast_alloc(
          struct ubr_advertised *a, char *string,
          struct ubr_interface *ifc,
          struct ubr_entry **new_entry)
{
     int rv;
     struct ubr_entry *c;

     /* This condition has no parameters so condition_string
      * should be empty or 0. */
     if ( !string || string[0] ) {
          if ( (c = kmalloc(sizeof(*c), GFP_ATOMIC)) ) {
               *new_entry = c;
               rv = 0;
          }
          else rv = -ENOMEM;
     }
     else rv = -EINVAL;

     return rv;
}
```

Slika 60: Razrez opisa in priprava pogoja.

Potrebujemo tudi funkcijo, ki bo na podlagi besedilnega opisa pripravila podatkovno strukturo. V opisanem primeru nimamo nobenih dodatnih parametrov. Tako ne pričakujemo praznega niza `condition_string`. Ker ne bomo hranili nobenih dodatnih parametrov, tudi ne

potrebujemo prostora zanje, zato zasežemo le toliko pomnilnika, kot ga potrebuje sama struktura `ubr_condition`. Polja `private_data` tako ne bo.

Ostalim delom strukture `ubr_condition` (`order`, `type` in `advertised`) ni treba nastavljati začetnih vrednosti. To bo za nas naredilo ogrodje in tako razbremenilo vtič. Na sliki 60 vidimo funkcijo `is_target_mac_broadcast_parse()`, ki opravlja našteto.

```
static int is_target_mac_broadcast_free(struct ubr_entry *entry)
{
     kfree(entry);
     return 0;
}
```

Slika 61: Sproščanje pogoja.

Funkcija, ki sprošča zasežene pogoje, je v našem primeru zelo preprosta. Pravzaprav bi jo lahko celo izpustili, v tem primeru pa bo infrastruktura našega mosta pogoj sprostila kar z uporabo funkcije `kfree()`. Ker gre za manjši trik, naš namen na tem mestu pa je prikaz uporabe, tega ne bomo naredili.

Funkcija `is_target_mac_broadcast_free()` ne naredi nič drugega, kot da sprosti pomnilnik, ki smo ga prej zasegli (slika 61).

```
static struct ubr_advertised is_target_mac_broadcast = {
     .module = THIS_MODULE,
     .name = "is_target_mac_broadcast",
     .description = "",
     .func = is_target_mac_broadcast_check,
     .alloc = is_target_mac_broadcast_alloc,
     .print = 0,
     .free = is_target_mac_broadcast_free
};

if ( ubr_advertised_register(&is_target_mac_broadcast) )
     printk(KERN_ERR "Unable to register condition '%s'.\n",
               target_mac_broadcast.name);

...

if ( is_target_mac_broadcast.use_count == 0 )
     ubr_advertised_unregister(&is_target_mac_broadcast);
```

Slika 62: Prijava in odjava pogoja.

Končno smo prišli do točke, ko lahko pogoj prijavimo jedru univerzalnega mosta. S tem bo pogoj na voljo vsem, ki bi ga želeli uporabljati.

Pogoj najprej opišemo s strukturo `ubr_advertised`, potem pa ga preko klica `ubr_advertised_register()` prijavimo jedru univerzalnega mosta. Pri tem je treba preveriti, ali je bila prijava uspešna. Zdaj je pogoj na voljo za uporabo. Na koncu pogoj tudi odjavimo, a le, če ni več v uporabi[54] (slika 62).

54 Klic funkcije `ubr_advertised_unregister()`, pri čemer bi bil pogoj še vedno v uporabi, ne naredi ničesar, vrne pa napako.

Ker naš pogoj nima nobenih zasebnih podatkov (ne uporablja polja `private_data`), tudi funkcija za njihov izpis ni potrebna. Zato postavimo vrednost `print` na `0`.

9.3.3.3 Primer prijave akcije

Pogoji in akcije so glede prijave in odjave v sistem zelo podobni. Prejšnji primer je bil zelo preprost. Pogoj ni zahteval nobenih dodatnih podatkov. Tokrat bo drugače. Dodali bomo akcijo, ki bo take dodatne podatke potrebovala.

```
static int set_type_len_func(struct ubr_frame *frame,
          struct ubr_entry *entry)
{
     unsigned short *type_len_ptr;

     type_len_ptr = (unsigned short *)(frame->skb->data +
     ETH_ALEN*2);
     *type_len_ptr = *((unsigned short *)entry->private_data);
}
```

Slika 63: Izvedba akcije.

Potrebujemo akcijo, ki nastavi polje `Type/Length` v okvirju Ethernet. Nova vrednost bo nastavljiva – torej mora biti nekje shranjena.

```
static int set_type_len_alloc(struct ubr_advertised *a,
          char *string, struct ubr_interface *ifc,
          struct ubr_entry **new_entry)
{
     int rv;
     struct ubr_entry *e;
     unsigned int new_type_len;

     if ( string && (1 == sscanf(string, "%d", &new_type_len))){
          /* Allocate memory and make sure we were successful. */
          if ( (e = kmalloc(sizeof(*e) + sizeof(unsigned short),
                    GFP_ATOMIC)) ) {
               *((unsigned short *)e->private_data) =
                         htons(new_type_len);
               *new_entry = e;
               rv = 0;
          }
          else rv = -ENOMEM;
     }
     else rv = -ETOOSMALL;
     return rv;
}
```

Slika 64: Razčlenitev opisa akcije in priprava akcije.

Najprej poglejmo funkcijo, ki bo akcijo izvedla. Polje `Type/Length` se nahaja v okvirju Ethernet tik za ciljnim in izvornim naslovom MAC. Kako pridemo do okvirja, smo videli pri pogoju. Tokrat se moramo premakniti za velikost dveh naslovov MAC naprej (to je `ETH_ALEN*2`). Preostanek funkcije je namenjen pretvarjanju podatkovnih tipov (slika 63).

Funkcija, ki iz oglaševane pripravi pravo akcijo, je tokrat bolj zapletena. Iz vhodnih parametrov moramo razbrati novo vrednost polja `Type/Length`, ob zaseganju pomnilnika pa moramo zaseči dva dodatna zloga, kamor bomo to vrednost shranili (slika 64).

Tokrat potrebujemo funkcijo za izpis zasebnih vrednosti. Za lažje branje bomo vrednost izpisali v šestnajstiški obliki (slika 65).

```
static int set_type_len_print(
              struct ubr_entry *entry,
              char *buffer,
              int buffer_size)
{
     return snprintf(buffer, "Type/Len=0x%04x",
              ntohs(*(unsigned short *)action->private_data));
}
```

Slika 65: Izvedba akcije.

Za konec pripravimo opis objavljene akcije in pokličemo funkcijo `ubr_register_action()`, da akcijo prijavimo jedru univerzalnega mosta (slika 66).

```
static struct ubr_advertised_action set_type_len = {
     .module = THIS_MODULE,
     .name = "set_type_len",
     .description = "<new_type_len>",
     .func = set_type_len_func,
     .alloc = set_type_len_alloc,
     .print = set_type_len_print,
     .free = 0
};

if ( ubr_advertised_register(&set_type_len) )
     printk(KERN_ERR "Unable to register action '%s'.\n",
              set_type_len.name);

...

if ( set_type_len.use_count == 0 )
     ubr_advertised_unregister(&set_type_len);
```

Slika 66: Prijava in odjava akcije.

9.3.3.4 Primer prijave razčlenjevalnika

Kot primer razčlenjevalnika bomo prikazali kar tistega, ki izpiše vse podprte ukaze, če navedemo, kateri ukaz nas zanima, pa njegov podrobnejši opis.

Najprej poglejmo sintakso ukaza:

```
? [<command>]
help [<command>]
```

Podprli bomo torej dva ukaza, oba s povsem enako sintakso, razlika bo le v ukazu samem. Naslednja stvar, ki jo potrebujemo, je funkcija, ki opravi razrez parametrov in izvedbo ustreznega ukaza.

```
static int ubr_parser_func_help(struct ubr_parser *parser,
          char *parms, char *result, int result_len)
{
     int rv = 0;
     struct hlist_node *n;
     struct ubr_parser *e;

     if (parms && *parms) {
          hlist_for_each_entry(e, n, &ubr_main.parser_prefixes,
                    hlist) {
               if (!strcmp(e->prefix, parms)) {
                    snprintf(result, result_len, "%s\n",
                              e->description);
                    return 0;
               }
          }
          printk("Command '%s' not supported.\n", parms);
     }
     else {
          int len = result_len;
          hlist_for_each_entry(e, n, &ubr_main.parser_prefixes,
                    hlist) {
               if ( (len -= snprintf(result + result_len - len,
                         len, "%s\n", e->prefix)) <= 0) {
                    rv = -ENOSPC;
                    break;
               }
          }
     }
     return rv;
}
```

Slika 67: Funkcija, ki opravlja razrez ukazov tabele naslovov MAC.

V našem primeru predstavlja prvi del, torej `?` in `help`, predpono ukaza. S prijavo razčlenjevalnika za to predpono dosežemo dvoje:

- vsi ukazi, ki se začnejo s to predpono, bodo preusmerjeni v naš razčlenjevalnik[55] in
- razčlenjevalnik bo dobil v obdelavo niz, ki sledi predponi.

Funkcija, ki opravi razrez ukaza,je prikazana na sliki 67.

Funkcijo imamo, zdaj pa potrebujemo še prijavo funkcije v jedro univerzalnega mosta. Najprej bomo definirali podatkovno strukturo, ki opisuje razčlenjevalnik. V našem primeru bomo definirali kar dva ukaza, `?` in `help`, zato potrebujemo tudi dve strukturi (slika 68).

55 Iz tega izhaja tudi, da ni mogoče prijaviti dveh razčlenjevalnikov za isto predpono.

```
static struct ubr_parser questionmark_parser = {
     .module      = THIS_MODULE,
     .prefix      = "?",
     .description = "[<command>]",
     .parser_func = ubr_parser_func_help
};
static struct ubr_parser help_parser = {
     .module      = THIS_MODULE,
     .prefix      = "help",
     .description = "[<command>]",
     .parser_func = ubr_parser_func_help
};
```

Slika 68: Podatkovna struktura za opis razčlenjevalnika.

S tema strukturama zdaj lahko opravimo prijavo in odjavo v jedro mosta. To opravi koda na sliki 69.

```
if ( ubr_parser_register(&questionmark_parser) )
     printk(KERN_ERR "Unable to register parser prefix '?'.\n");
if ( ubr_parser_register(&help_parser) )
     printk(KERN_ERR "Unable to register parser prefix
     'help'.\n");

...

ubr_parser_unregister(&questionmark_parser);
ubr_parser_unregister(&help_parser);
```

Slika 69: Prijava in odjava predpone.

9.3.4 Programski vmesnik, namenjen uporabi mosta

Programski vmesnik za uporabo mosta je zelo preprost – sestavlja ga ena sama funkcija (slika 70).

```
int ubr_parser_execute(char *command, char *result, int
     result_len);
```

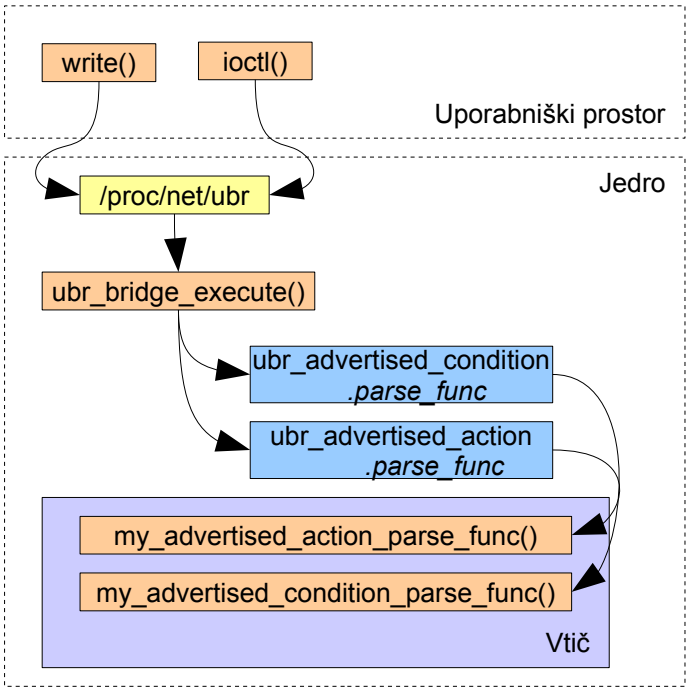
Slika 70: Programski vmesnik za uporabo mosta.

Izbira programskega vmesnika za uporabo mosta je ena težavnejših. Po eni strani je izbira na besedilu osnovanega vmesnika zelo prilagodljiva, še posebej v kombinaciji z možnostjo prijavljanja novih ukazov (to smo podprli z možnostjo prijave razčlenjevalnikov za poljubne predpone). Po drugi strani pa je tak vmesnik tudi okoren, saj je treba podatke vedno znova pretvarjati v besedilo in nazaj, ob tem pa je obilo možnosti za napake.

V našem primeru je na koncu prevladala prilagodljivost. Pri tem igra pomembno vlogo tudi dejstvo, da pri mostu vmesnik za upravljanje ni kritičen s stališča porabe sistemskih virov. V primerjavi z obdelovanjem okvirja so upravljavski posegi razmeroma redki.

Vmesnik univerzalnega mosta omogoča dostop aplikacijam iz uporabniškega naslovnega prostora, ki uporabljajo sistemska klica `write()` in `ioctl()`. Drugim moduli v Linuxovem jedru lahko funkcijo kličejo neposredno (slika 71).

Klice, ki se nanašajo na dodajanje pogojev in akcij – v teh primerih je treba razbrati parametre pogoja ali akcije –, univerzalni most preda vtičnemu modulu, ki je pogoj ali akcijo objavil. To naredi tako, da pokliče funkcijo, ki smo jo ob registraciji zapisali v polje `parse_func`.



Slika 71: Uporaba vmesnika univerzalnega mosta.

9.3.4.1 Osnovni ukazi univerzalnega mosta

Doslej predstavljeni mehanizem za razširjanje ukazne vrstice univerzalnega mosta uporablja tudi most sam. Ta prijavi predpone (`?`, `help`, `bridge`, `port` in `rule`), s pomočjo katerih lahko upravljamo most. Osnovni ukazi, ki so podprti na ta način, so prikazani na sliki 72.

Ko posebnost naj omenimo, da je pri dodajanju mogoče določiti tudi negacijo pravila. To omogoča izgradnjo krajših in bolj čitljivih pogojev. Negacija pogoja je izvedena na nivoju infrastrukture, zato je ni treba implementirati v vsako pogojno funkcij posebej. Tako so tudi funkcije, ki preverjajo pogoje, lahko krajše.

Ker beseda `not` označuje negacijo pravila, je njena uporaba v imenih pravil in akcij prepovedana. Negacija akcije nima nobenega učinka.

9.3.4.1.1 Splošni podatki

Ukaze, ki so na voljo, dobimo s pomočjo ukazov `help` in `?`.

Do splošnih podatkov pridemo s pomočjo ukaza `show`, ki mu sledi ime domene, ki nas zanima.

9.3.4.1.2 Delo z mostovi

Nov most dodamo z ukazom `bridge add`. Obvezno mu dodamo še ime, ki ga bo dobil novi most.

Brisanje mosta opravimo z ukazom `bridge del`.

9.3.4.1.3 Delo z napravami (vmesniki)

Napravo dodamo mostu z ukazom `interface add`, z ukazom `interface del` pa jo z mosta odvzamemo.

9.3.4.1.4 Delo s seznami za vhodno in izhodno obdelavo

Nov pogoj, akcijo ali oznako dodamo v seznam z ukazom `rule add`. Parametri, ki sledijo, povedo, na katero napravo se nahaja ukaz, poleg tega pa še smer (vhodna ali izhodna obdelava), kaj dodajamo, ime pogoja, akcije ali oznake in še njen položaj v spisku. Slednji je lahko absoluten (recimo 100), lahko pa relativen (pred/za, na začetek/konec).

Brisanje poteka s pomočjo ukaza `rule del`, pri čemer moramo zopet povedati, za katero napravo gre, poleg tega pa še smer in indeks.

```
? [<command>]
help [<command>]
bridge add <bridge_name> [<mac_table_manager>]
bridge del <bridge_name>
interface add <bridge_name> <device_name>
interface del <device_name>
rule add <device_name> <direction> <type> ['not'] <name>
     <position> [<params>]
rule del <device_name> <direction> <position>
     bridge_name = Name of bridge device.
     device_name = Name of network device.
     direction = 'ingress' | 'egress'
     type = 'condition' | 'action' | 'label'
     name = Valid name of a published condition or action.
     position = 'head' | 'tail' | 'before' <index_or_label> |
     'after' <index_or_label> | <index>
     index_or_label = <index> | Valid label name.
     index = Valid integer.
     params = Any string that condition or action knows how to
     parse.
```

Slika 72: Osnovni ukazi univerzalnega mosta.

10 Primeri uporabe

V tem poglavju bomo pogledali, kako nastavimo most, da podpira razne funkcionalnosti, ki smo jih opisali v prejšnjem poglavju. Iz prejšnjega poglavja si bomo izposodili tudi jezik za opis pogojev in akcij.

Naš glavni namen je pokazati, da je mogoče tudi z zelo omejenim naborom pogojev in akcij podpreti širok nabor funkcionalnosti.

10.1 Osnovni pogoji

Pogoji se dotikajo lastnosti okvirja na različnih nivojih:

- L1 (vhodna in izhodna naprava);
- L2 (izvorni in ciljni naslov MAC, delo z označbami VLAN);
- L3 (protokola IPv4 in IPv6);
- L4 (protokoli TCP, UDP, ICMP, IGMP);
- drugo.

Težava pri vnaprejšnji definiciji pogojev je v zadnji točki – drugo. Težko vnaprej predvidimo vse možne pogoje, po katerih bomo želeli iskati okvirje.

10.1.1 Naprava

Pri preverjanju vmesnika nas vedno zanima le vhodna naprava. Razlog je preprost. V času obdelave vhodnih seznamov točno vemo, na kateri vhodni napravi se nahajamo, o izhodni pa še nič. V času izhodne obdelave vemo, na katerih izhodni napravi smo, kot pogoj pa lahko preverimo, iz katere vhodne naprave smo prišli do sem. Tako lahko na izhodu drugače obravnavamo promet, ki ga je most prejel tako na eni napravi kot na drugi napravi.

10.1.2 Odmik, maska in vrednost

Eden od možnih izhodov, sicer precej neprijazen, je pogoj, pri katerem uporabimo kombinacijo odmika, maske in vrednosti. Na določenem odmiku od začetka okvirja uporabimo masko in preverimo, če se rezultat ujema z vrednostjo.

Tak način je zelo priljubljen pri stikalih, ki so večinoma ali v celoti izdelana strojno, razlog pa je očiten. Strojna izvedba takega primerjanja je zelo učinkovita, hkrati pa preprosta. Z uporabo takih pogojev je mogoče simulirati praktično vse druge pogoje, a ne vedno na najbolj učinkovit način. Poglejmo dva primera, ki bosta prikazala moč in nemoč takih pogojev.

10.1.2.1 Ujemanje VLAN-ov

Zelo pogost pogoj se nanaša na ujemanje oznake VLAN. Zanima nas:

1. ali okvir vsebuje značko VLAN in
2. ali ima značka VLAN (ali del nje) določeno vrednost.

Oboje razmeroma lahko preverimo, če vemo da:

- imajo paketi, ki vsebujejo značko VLAN, na odmiku 13-14 od začetka okvirja shranjen tip paketa, ki mora biti `0x8100`, in
- tipu paketa takoj sledita dva zloga, od katerih spodnjih 12 bitov pomeni številko VLAN-a.

Pogoj lahko torej povzamemo takole:

Zlog	1	2		12	13	14	15	16	17	
Maska	0x00	0x00	…	0x00	0xff	0xff	0x0f	0xff	0x00	…
Vrednost	0x00	0x00		0x00	0x81	0x00	0x0y	0xyy	0x00	

Ker nas zlogi pred in za masko ne zanimajo, lahko to zapišemo tudi krajše, s trojčkom `[odmik, maska, vrednost]` (`yyy` označuje vrednost značke VLAN, ki jo iščemo):

```
Frame <offset> <mask> <value>
```

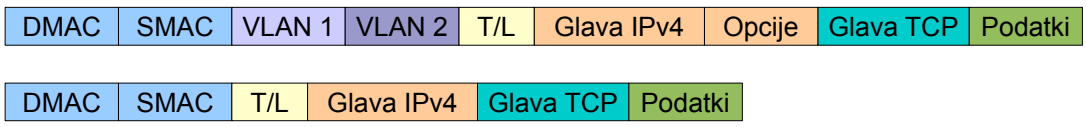
```
Frame 13 0xffff0fff 0x81000yyy
```

Strojne izvedbe imajo običajno omejitve glede dolžine maske in vrednosti, pri programski izvedbi pa nas to ne skrbi. Ujemanje takega pogoja znamo seveda zelo preprosto preveriti. V primeru iskanja z določeno značko VLAN označenega paketa se tak pristop izkaže dokaj dobro.

V primerih, ko je odmik iskanega podatka od začetka okvirja vnaprej znan, je uporaba tega pristopa zelo priročna. Tako lahko brez večjih težav pokrijemo vse poizvedbe po izvornem in ciljnem naslovu MAC, tipu paketa in tudi znački VLAN. Že pri drugi znački VLAN pa lahko nastopi nekaj težav. Naslednji primer bo te težave dobro prikazal.

10.1.2.2 Ujemanje ciljnih vrat TCP

Ujemanje izvornih vrat TCP je povsem drugačen problem. Denimo, da želimo s pogojem preveriti, ali gre za okvir, ki je del podatkovnega toka protokola HTTP. Zanje je značilno, da gre za okvirje, ki pripadajo protokoloma IPv4 ali IPv6, znotraj tega pa protokolu TCP, pri katerem imajo ciljna vrata številko `80`. Na sliki 73 vidimo obe skrajni možnosti, ki vodita od začetka okvirja do mesta, kjer lahko končno izvedemo primerjavo številke vrat. Številka ciljnih vrat je zapisana v 3. in 4. zlogu glave TCP.



Slika 73: Obe skrajni možnosti dostopa do glave TCP.

Če želimo z uporabo odmik-maska-vrednost ujeti tak okvir, potem moramo definirati množico pravil za vse možne primere:

- nič, ena ali dve znački VLAN[56], pri čemer tudi tip značke VLAN (angl. *Ethernet type* ali skrajšano *EtherType*) ni nujno `0x8100`, pač pa lahko tudi kaj drugega.
- Glava IPv4 lahko vsebuje različno število opcij (vrednost polja IHL v glavi IPv4 je lahko med 5 in 15).

Tudi brez možnosti različnih tipov značke VLAN imamo opravka s precejšnjim številom možnosti – kar 33 (3 možnosti značk VLAN, 11 možnosti pri velikosti glave IPv4). Če želimo uporabiti filtriranje odmik-maska-vrednost, potem ne gre drugače, kot da navedemo 33 pogojev. To seveda ni praktično, pa tudi časovno potratno je.

10.1.3 Tip vsebovanega prometa

Pri prejšnjem pogoju smo pokazali, kako nerodno lahko postane preverjanje vsebine okvirja, ko enkrat ne moremo več vnaprej predvideti natančnega odmika od začetka okvirja, ali pa je možnosti preveč.

Z možnostjo preverjanja tipa vsebovanega prometa (angl. *Ethernet type*) bomo močno olajšali izbiranje določene vrste prometa. Ta pogoj bo znal „preskočiti“ poljubno število značk VLAN, ki bodo poleg tega lahko tudi različnih tipov. Tako bomo denimo paket ARP lahko poiskali preprosto tako, da bomo zahtevali, da je tip vsebovanega prometa `0x0806` ([43]):

```
EtherType <value>
EtherType 0x0806
```

10.1.4 Tip, odmik, maska, vrednost

V primeru iz poglavja 10.1.2.2 smo imeli težave s spremenljivim številom značk VLAN. Vseeno pa pristop odmik-maska-vrednost ni bil slab, vsaj s stališča splošnosti ne. Smiselno je torej, da uvedemo akcijo, ki za določen tip prometa uporabi prej omenjeni pristop. Na tak način lahko zelo enostavno opravimo preverjanja znotraj paketov ARP, glav IPv4 in IPv6, pa še precej drugih.

Naslednji filter bi prepoznal pakete IGMP, ki prihajajo z naslova `1.2.3.4`:

```
Payload <ether_type> <offset> <mask> <value>
Payload 0x0800 10 0xff0000ffffffff 0x02000001020304
```

Zahtevamo, da je tip vsebovanega prometa IPv4 (0x0800), potem pa nas zanima 10. zlog, ki sledi (v paketu IPv4 vsebovani protokol), njegova vrednost mora biti `2` (IGMP), potem pa poskrbimo še, da je izvorni naslov IPv4 poslanih paketov `1.2.3.4`.

10.1.5 IP odmik, maska, vrednost

Naslednjo težavo v primeru iz poglavja 10.1.2.2 je predstavljala spremenljiva velikost glave IPv4. Protokol IPv6 ima sicer glavo fiksne velikosti, a ji tudi lahko sledi še več glav. V obeh primerih to predstavlja težavo.

Lahko pa uvedemo nov tip pogoja, ki bo preskočil glavi IPv4 in IPv6 – na katerokoli pač

56 Tudi v praksi že obstajajo primeri, ko so v uporabi več kot dve znački VLAN. Če želimo ujeti tudi take primere, potem moramo pogoj ustrezno razširiti.

naleti –, in nadaljeval z delom na vsebini paketa IP. Na ta način nas elegantno prestavi na začetek glave IGMP, ICMP, TCP, UDP ali kakšnega drugega v paketu IP vsebovanega protokola.

Pogoj, ki preveri, ali imamo opravka s paketom HTTP (paket TCP s ciljnimi vrati `80`), je potem sledeč:

```
Ip <ip_protocol> <offset> <mask> <value>
Ip 6 3 0xffff 80
```

Preverjamo torej, če gre za paket IPv4 ali IPv6, v katerem je vsebovan protokol TCP (6, glej [44]), in če je, potem preverimo vrednost ciljnih vrat (3. in 4. zlog).

10.1.6 Vrednost števca

Za izvedbo nekaterih funkcionalnosti, na primer omejevanja prometa, nujno potrebujemo števce in pogoje, ki so od teh števcev odvisni.

```
Limit <counter> <value>
Limit dlf 10
```

10.2 Osnovne akcije

Pri pogojih smo uspeli narediti zelo veliko z majhnim številom razmeroma splošnih pogojev.

10.2.1 Odmetavanje okvirja

Predstavljena arhitektura ima svoje korenine v nadzoru dostopa (angl. *access control*). Najbolj tipična akcija tam je odmetavanje okvirja, če ta ne izpolnjuje določenih pogojev. Tako bo tudi prva akcija, ki jo bomo mi predstavili, odmetavanje okvirja.

```
Drop
```

10.2.2 Izpis okvirja

Predvsem med razvojem in testiranjem funkcionalnosti nam pride zelo prav možnost, da lahko izpišemo okvir. Toliko bolje, če lahko to naredimo na različnih točkah med obdelavo in tako spremljamo nastale spremembe.

```
Print
```

V kombinaciji s pogoji postane to zelo močno orodje tudi med delovanjem, saj lahko izpisujemo samo točno določeno vrsto prometa in tako med delovanjem spremljamo dogajanje[57].

10.2.3 Sledenje obdelavi

Druga zelo koristna možnost, še posebej pri odpravljanju napak, je sledenje obdelavi. V tem

57 Izpisovanje velikih količin podatkov je lahko precej potratno in lahko precej vpliva na zmogljivost mosta. Zaradi tega je možnost izpisa le določene vrste prometa zelo dobrodošla.

primeru sledimo poti okvirja skozi vhodno in izhodno obdelavo in tako spremljamo korake, ki jih pri obdelavi opravi most. Na ta način z lahkoto odkrijemo napake pri nastavitvah mosta.

```
Debug <state>
```

```
Debug 1
```

Podobno kot izpisovanje okvirjev ima tudi sledenje lahko precejšen vpliv na zmogljivost mosta, zato ga je smiselno uporabiti v kombinaciji s pogoji.

10.2.4 Brezpogojni skok

V poglavju 9.3.1.5 smo predstavili postopek obdelave pogojev in akcij v vhodnih in izhodnih vrstah. Videli smo, da zaporedje pogojev obravnavamo, kot bi bil med njimi logični *in*. Kako torej dosežemo učinek logičnega *ali*? S pomočjo brezpogojnega skoka to ni težko. Pravilo

```
IF a OR b THEN c
```

ob pomoči brezpogojnega skoka zapišemo takole:

```
    IF a THEN goto @1
    IF b THEN goto @1
    goto @2
@1
    c
@2
```

Če upoštevamo De Morganovi pravili, potem lahko vse še skrajšamo:

```
    IF !a AND !n THEN goto @2
    c
@2
```

V naših vhodno-izhodnih spiskih bi to videli takole:

```
1    not a
2    not b
3    goto 5
4    c
```

Očitno pa za izvedbo tega potrebujemo skok. Ker smo pogoje že predstavili v poglavju 10.1 in ne želimo podvajati funkcionalnosti, bomo sedaj uvedli le brezpogojni skok.

Po skoku se izvajanje v vhodnem ali izhodnem seznamu nadaljuje pri vnosu z indeksom, na katerega kaže skok. Če takega vnosa ni, potem nadaljujemo z izvajanjem pri prvem naslednjem vnosu. Če tudi tega ni, torej je preostanek seznama prazen, pa okvir:

- če smo obdelovali seznam vhodnih pravil, razpošljemo na vse ostale naprave, ki so priključene na most;
- če smo obdelovali seznam izhodnih pravil, odpošljemo okvir skozi napravo, katere izhodni seznam smo obdelovali.

Sintaksa akcije je preprosta:

```
Goto [<dev> <direction>] <index>
```

```
Goto 10
```

10.2.5 Pošiljanje okvirja

Sliši se skoraj neumno, a pošiljanje okvirja kot posebna akcija ima svoj smisel:

- operaterji želijo včasih preveriti promet z določene naprave in v ta namen vzpostavijo zrcaljenje naprave (angl. *port mirroring*), kar pomeni da se kopije vseh okvirjev, ki prihajajo z ali so usmerjeni na določen vmesnik, pošiljajo še nekam drugam;
- včasih želimo isto le za določeno vrsto prometa in tako pridemo do povsem nove storitve[58].

Pošiljanje okvirja na določen vmesnik postane še dosti bolj uporabno, če ob tem dovolimo skok na določeno mesto v izhodni ali vhodni vrsti.

```
Send <dev> [<direction> <index>]
Send eth0
Send dsl2 ingress 0
```

Seveda ni nobenega razloga, da akcije ne bi mogli uporabiti večkrat zapored. Na ta način lahko isti okvir pošljemo na več naprav.

10.2.6 Spreminjanje okvirja in odmik-maska-vrednost

Pristop, ki se nam je odlično služil, ko smo definirali pogoje, lahko uporabimo tudi pri spreminjanju okvirjev. S pomočjo odmika, maske in vrednosti bomo spreminjali poljubne podatke v okvirju po pravilu:

```
new_value = (old_value AND mask) OR value
```

Ukazi za spreminjanje delov okvirja pa so:

```
FrameSet <offset> <mask> <value>
PayloadSet <offset> <mask> <value>
IpSet <offset> <mask> <value>
FrameSet 15 0x1fff 0x2000
PayloadSet 2 0xf6 0x28
IpSet 3 0x0000 8800
```

Prvi ukaz nastavi vrednost polja CoS v znački VLAN na `1` (polje CoS je shranjeno v zgornjih 3 bitih značke VLAN), preostanka pa ne spreminja.

Drugi ukaz nastavi polje DSCP v glavi paketa IPv4 na `10` (polje DSCP je shranjeno v zgornjih 6 bitih 2. zloga glave IPv4).

Tretji ukaz nastavi ciljna vrata paketa TCP ali UDP na `8080` (ciljna vrata so shranjena v 3. in 4. zlogu glave TCP ali UDP).

10.2.7 Dodajanje v okvir

Pri dodajanju v okvir maske ne potrebujemo. Vedeti moramo le, kam in kaj želimo dodati.

58 Taka storitev je *IGMP telemetry log*, omogoča pa, da lahko nekdo, ne nujno ponudnik TV-vsebin, spremlja priljubljenost različnih TV-programov, kadar so ti posredovani preko IP TV.

Zopet pa zaradi enostavnejše uporabe predstavljamo tri različne funkcije, vsaka od njih je prirejena za delo z določenim delom okvirja:

```
FrameInsert <offset> <value>
PayloadInsert <offset> <value>
IpInsert <offset> <value>
FrameInsert 13 0x81000009
```

Prvi ukaz doda značko VLAN (Ethernet tip `0x8100`, VLAN ID `9`). Primera uporabe za drugi in tretji ukaz bi bila preveč zapletena, zato ju na tem mestu ne bomo obravnavali.

10.2.8 Brisanje iz okvirja

Tudi za brisanje podatkov iz okvirja bomo predstavili podobno akcijo kot za spreminjanje in dodajanje. Potrebujemo le podatek o začetku in dolžini izbrisa:

```
FrameDel <offset> <length>
PayloadDel <offset> <length>
IpDel <offset> <length>
FrameDel 13 4
```

Ukaz v primeru izbriše značko VLAN (dolga je 4 zloge).

10.2.9 Kontrolne vsote

Mnogi protokoli preverjajo pravilnost podatkov s pomočjo kontrolnih vsot. Protokol IPv4 vsebuje kontrolno vsoto, ki preverja pravilnost podatkov v glavi. Protokoli TCP, UDP in IGMP imajo kontrolne vsote, ki preverjajo pravilnost glave in podatkovnega dela. Protokola ICMP in ICMPv6 s kontrolno vsoto preverjata pravilnost glave.

Po spremembi katerega od teh delov s pomočjo akcij iz poglavij 10.2.5 , 10.2.6 in 10.2.7 moramo ponovno izračunati kontrolne vsote, v nasprotnem primeru bodo prejemniki okvirje razumeli kot okvirje z napako in jih zavrgli. Zaradi tega uvajamo akcijo, ki zna po izvedenih spremembah osvežiti prej naštete tipe kontrolnih vsot:

```
UpdateChecksums
```

10.2.10 Tabela naslovov MAC

Omenili smo že, da bomo funkcionalnost tabele naslovov MAC ponudili v obliki akcij, ki jih lahko uporabimo po želji.

Delo tabele lahko razdelimo v dva dela: učenje in posredovanje. V prvi fazi se učimo, na kateri napravi smo prejeli okvirje z in njihove izvorne naslove MAC. V drugi okvirje usmerjamo proti ciljnim napravam na podlagi njihovega ciljnega naslova MAC in znanja, ki smo ga zbrali v fazi učenja.

```
MacTableLearn
MacTableForward
```

10.2.11 Merjenje prometa

Akcije, ki omogočajo merjenje prometa, so zelo zanimive, v kombinaciji s pogoji, ki se nanašajo na števce, pa omogočajo omejevanje prometa.

Števce bomo ločevali glede na dve lastnosti, skupno bomo tako imeli štiri vrste števcev:

- Čas merjenja:
  - *neprestano* – števec se neprestano povečuje;
  - *intervalno* – števec se povečuje, po določenem intervalu pa se sam postavi na 0.
- Predmet merjenja:
  - *okvir* – enota merjenja je okvir;
  - *zlogi* – enota merjenja so zlogi, ki sestavljajo okvir.

```
Set <counter> <value>
Count [Rate] [Size] <counter>
Count Rate dlf
Count Size igmp
Set dlf 0
```

Prvi primer šteje okvirje na sekundo. Drugi primer šteje prenesene zloge. Zadnji primer postavi vrednost števca `dlf` na vrednost `0`.

10.3 Primeri postavitev

V tem poglavju bomo pogledali primere postavitev. Pri tem bomo uporabili pogoje in akcije, ki smo jih doslej predstavili.

10.3.1 Dodajanje značke VLAN

Na vmesniku `dsl0` bomo na vhodu dodali značko VLAN tipa `0x8100`[59]. Okvir bo spadal v VLAN `10`, polji prioriteta in CFI bosta `0`.

```
rule add dsl0 ingress 0 action FrameInsert 13 0x8100000a
```

10.3.2 Brisanje značke VLAN za izbrane VLAN-e

Na vmesniku `dsl2` bomo na izhodu brisali zunanje glave VLAN, če so te tipa `0x88a8`, VID pa je `15` ali `20`.

```
rule add dsl2 egress 0 condition not Frame 13 0xffff 0x88a8
rule add dsl2 egress 1 action goto 7
rule add dsl2 egress 2 condition Frame 15 0x0fff 0x000f
```

59 Prvotni standard IEEE 802.1Q je predvideval le glave VLAN tipa `0x8100`, kasneje pa je standard IEEE 802.1ad predvidel še druge tipe glave. Trenutno je standardizirana tudi vrednost `0x88a8`, različna stikala pa uporabljajo tudi druge vrednosti. Zaradi tega moramo dovoliti nastavljanje tipa glave VLAN.

```
rule add dsl2 egress 3 action goto 6
rule add dsl2 egress 4 condition not Frame 15 0x0fff 0x0014
rule add dsl2 egress 5 action goto 7
rule add dsl2 egress 6 action FrameDel 13 4
```

Kot vidimo, moramo zaradi pravila, ki vključuje logični *ali*, skozi kar nekaj zapletov.

10.3.3 Prestavljanje iz enega v drug VLAN

Na vmesniku `eth0` bomo vhodni promet iz VLAN-a `20` prestavili v VLAN `5`.

```
rule add eth0 ingress 0 condition Frame 13 0xffff0fff 0x81000014
rule add eth0 ingress 1 action FrameSet 15 0x0fff 0x0005
```

10.3.4 Dodajanje značke VLAN v odvisnosti od vrste prometa

Paketom IGMP bomo na vhodu na vmesniku `dsl10` dodali značko VLAN tipa `0x8100` z VLAN ID-jem `1000`.

```
rule add dsl10 ingress 0 condition Payload 0x0800 10 0xff 2
rule add dsl10 ingress 1 action FrameInsert 13 0x810003e8
```

10.3.5 Nastavljanje prioritete

Paketom IGMP iz prejšnje alineje bomo nastavili prioriteto `3`.

```
rule add dsl10 ingress 2 action FrameSet 15 0x1f 0x60
```

10.3.6 Omejevanje glede na VLAN

Na vmesniku `dsl2` bomo sprejemali samo promet brez značk VLAN. Na vmesniku `eth0` bomo sprejemali le promet, označen z značkami VLAN tipa `0x88a8`. Na vmesniku `dsl2` hkrati tudi ne sme promet iz VLAN-a `5`.

```
rule add dsl2 ingress 0 condition Frame 13 0xffff 0x8100
rule add dsl2 ingress 1 action Drop
rule add eth0 ingress 0 condition not Frame 13 0xffff 0x88a8
rule add eth0 ingress 1 action Drop
rule add dsl2 egress 0 condition Frame 13 0xffff0fff 0x81000005
rule add dsl2 egress 1 action Drop
```

10.3.7 Omejevanje glede na vrsto prometa

Na vmesniku `dsl7` bomo dovolili samo sprejem okvirje PPPoE. To pomeni, da moramo prepuščati okvirje PPPoED in PPPoES.

```
rule add dsl7 ingress 0 condition EtherType 0x8863
rule add dsl7 ingress 1 action Goto 5
rule add dsl7 ingress 2 condition EtherType 0x8864
rule add dsl7 ingress 3 action Goto 5
rule add dsl7 ingress 4 action Drop
```

10.3.8 Omejevanje glede na količino prometa

Na vmesniku `eth0` bomo na sekundo sprejeli do 100 paketov ICMPv4:

```
rule add eth0 ingress 0 condition not Payload 0x0800 10 1
rule add eth0 ingress 1 action Goto 5
rule add eth0 ingress 2 action Count Rate eth0_icmp
rule add eth0 ingress 3 condition not Limit eth0_icmp 100
rule add eth0 ingress 4 action Drop
```

10.3.9 Sledenje izvajanja

V nekaterih primerih je zelo priročno, če lahko za en sam okvir opazujemo, kaj se z njim dogaja ob prehajanju mosta. Drugič nas zanima prehajanje vseh okvirjev določene vrste.

10.3.9.1 Sledenje enemu samemu okvirju med vhodno obdelavo

Na vmesniku `dsl5` bomo na vhodu sledili dogajanju z enim okvirjem prometa, ki prihaja po VLAN-u `10`.

```
rule add dsl5 ingress 0 condition Frame 13 0xffff0fff 0x8100000a
rule add dsl5 ingress 1 action Debug 1
...
rule add dsl5 ingress tail action Debug 0
```

10.3.9.2 Sledenje izhodni obdelavi določenega tipa okvirjev

Zanima nas, kaj se dogaja z okvirji PPPoED, ki se na vmesniku `eth0` občasno izgubljajo v izhodni smeri.

```
rule add eth0 egress 0 condition EtherType 0x8863
rule add eth0 egress 1 action Debug 1
...
rule add eth0 egress tail action Debug 0
```

10.3.10 Ločevanje uporabniških vmesnikov

Imamo most s tremi vmesniki. `dsl0` in `dsl1` sta namenjena priključevanju uporabnikov, `eth0` pa povezovanju v omrežje. Promet iz obeh uporabniških vmesnikov sme samo na `eth0` (angl. *port security*), v obratni smeri pa želimo promet usmerjati na podlagi tabele naslovov MAC. Če to ne uspe, potem okvir pošljemo obema uporabniškima vmesnikoma.

```
rule add eth0 ingress 0 action MacTableForward

rule add dsl0 ingress 0 action MacTableLearn
rule add dsl0 ingress 1 action Goto eth0 egress 0

rule add dsl1 ingress 0 action MacTableLearn
rule add dsl1 ingress 1 action Goto eth0 egress 0
```

10.3.11 Izvedba dela protokola CFM

Povsem pravilna izvedba protokola CFM [8] je na klasičnih mostovih praktično nemogoča. Razlog za to so zahteve, ki jih postavlja standard CFM – pakete je treba prestreči in vračati v tok v vhodnih in izhodnih obdelovalnih vrstah. Z doslej predstavljenimi pogoji in akcijami lahko poskrbimo, da bomo pakete prestrezali, jih prepustili v obdelavo aplikaciji v

uporabniškem naslovnem prostoru, potem pa jih zopet vrnili v obdelavo tik za mestom, na katerem smo jih prestregli.

```
local add cfm_eth0 cfm_eth0_us
bridge add br0
interface add br0 cfm_eth0
...
rule add eth0 ingress 0 condition Frame 13 0xffff 0x8902
rule add eth0 ingress 1 action Goto cfm_eth0 egress 0
rule add cfm_eth0 ingress 0 action Goto eth0 ingress 2
```

Aplikacija bo okvirje sprejemala na napravi `cfm_eth0_us` (oznaka `us` v tem primeru pomeni uporabniški prostor, angl. *user space*), ta pa bo le lokalna preslikava naprave `cfm_eth0`, ki jo bomo pripeli na most `br0`. Na napravi `cfm_eth0_us` oddani okvirji bodo za most vidni, kot da prihajajo z naprave `cfm_eth0`, tam pa jih bo vhodna obdelava prestavila na mesto 2 v vhodni vrsti naprave `eth0`. Most bo nadaljeval obdelavo tako, kot da je bil okvir dejansko prejet na napravi `eth0`.

10.3.12 Spremljanje prometa IGMP

Spremljanje prometa IGMP (angl. *telemetry log*) pomeni, da vse pakete IGMP zrcalimo na izbrano napravo. Mi bomo pakete IGMP zrcalili na napravo `eth1`.

```
rule add eth0 egress 0 condition Payload 0x0800 10 2
rule add eth0 egress 1 action Send eth1
```

11 Primer izvedbe

Da bi lažje preverili praktično izvedljivost nekaterih idej in preverili nekatere arhitekturne pristope, smo se odločili pripraviti tudi izvedbo univerzalnega mosta za operacijski sistem Linux.

11.1 Zakaj Linux

Izbira, da primer izvedbe pripravimo za operacijski sistem Linux, je bila preprosta. K temu so nas navedli odprta arhitektura omrežnega dela, odprtost izvorne kode in razvojnega okolja, obstoječe znanje in dostopnost primerov.

11.1.1 Odprta arhitektura omrežnega dela

Arhitektura omrežnega dela operacijskega sistema Linux je zelo odprta. Zelo preprosto omogoča dodajanje podpore za nove vrste strojnih in logičnih omrežnih naprav, za nove omrežne protokole itd. V veliki meri je bila prav ta odprtost tudi navdih pri načrtovanju univerzalnega mosta. Zaradi tega je izbira Linuxa za izvedbo povsem naravna.

Izvedba omrežnega dela jedra Linuxa dokaj dobro sovpada z idejami, ki sta jih Hutchinson in Peterson [24] promovirala pred dvajsetimi leti: da je treba omrežni del operacijskega sistema nekako formalizirati in na ta način poenostaviti izvedbo in uporabo omrežnih protokolov in naprav.

11.1.2 Odprtost izvorne kode in razvojnega okolja

Pomembna želja avtorjev je bila, da je izvedba prosta, dostopna vsem kot učni primer, obenem pa morajo biti tudi orodja za delo prosto na voljo. Na ta način je brez nepotrebnih stroškov omogočen poln dostop vsakomur, ki ima željo ali potrebo po delu s tovrstnimi rešitvami.

11.1.3 Obstoječe znanje

Seveda je imelo precejšen vpliv na izbiro tudi dejstvo, da razpolagamo z veliko izkušnjami z razvojem in delom na operacijskem sistemu Linux. Te izkušnje segajo od namiznih rešitev, strežnikov pa do vgradnih (angl. *embedded*) naprav.

Podobnih izkušenj iz drugih okolij (Network Simulator [34]) nimamo.

11.1.4 Dostopnost primerov

Ne glede na že pridobljeno znanje je vedno dobrodošlo, če se lahko ob težavah obrneš na ljudi, ki ti lahko pomagajo ali pa si pomagaš kar sam, pri tem pa uporabiš že delujoče primere. Operacijski sistem Linux je v tem pogledu en sam velik učni laboratorij.

11.2 Razvojno okolje

Razvoj je potekal v okolju distribucije[60] Debian [49], ki smo jo namestili v navidezni računalnik, ustvarjen s pomočjo orodja VMware Server [50]. VMware ni edino orodje za pripravo navideznih računalnikov, a se je pri delu izkazalo bolje (hitrejše, manjša poraba virov, nekatere priročne malenkosti), kot povsem odprti VirtualBox [51]. Drugih tovrstnih orodij nismo preizkušali.

Uporaba navideznega računalnika seveda ni obvezna, je pa zelo priročna. Poleg dejstva, da nam prostora fizično ne zaseda še en računalnik, je najbolj koristno to, da je ponovni zagon zelo priročen[61]. To je še posebej pomembno, kadar med delom nismo fizično ob računalniku, ampak od njega oddaljeni.

Druga koristna lastnost navideznih računalnikov je, da jim je mogoče preprosto spreminjati strojno opremo[62]. Tako lahko zelo hitro dodamo več omrežnih naprav, na tak način pa je testiranje lažje. Ko smo sestavljanja navideznih računalnikov dovolj vešči, lahko v namiznem računalniku zgradimo celo omrežje navideznih računalnikov, to pa močno poenostavi testiranje.

Kljub temu, da je urejevalnik `vi` na voljo na vseh operacijskih sistemih, ki spadajo v družino operacijskih sistemov UNIX, pa je delo z večjimi kosi izvorne kode vseeno mnogo lažje s katerim od sodobnih razvojnih okolij (angl. *Integrated Development Environment*, IDE). Mi smo izbrali orodje Eclipse [52], ki med drugim ponuja tudi vtič za razvoj v programskih jezikih C in C++.

Univerzalni most smo razvili za jedro operacijskega sistema Linux z oznako 2.6.37 [41], danes pa so na voljo že novejše različice. Med različicami so manjše razlike, ki pa občasno privedejo do tega, da je treba izvorno kodo vseeno prilagoditi.

11.3 Univerzalni most in jedro Linux

Da lahko modul uporabimo v jedru operacijskega sistema Linux, ga moramo zanj ali vanj prevesti. Prvo pomeni, da ga prevedemo kot ločen modul, ki ga kasneje naložimo in s tem postane del jedra, drugo pa, da modul že v času prevajanja postane neločljiv del jedra. Med delovanjem ga lahko še vedno nadzorujemo preko različnih nastavitev, ne moremo pa ga iz jedra odstraniti.

11.3.1 Vključitev v prevajanje

V prevajanje jedra Linux vključimo univerzalni most na enak način kot vsako drugo komponento:

1. Pripravimo opis komponente v datoteki `Kconfig` (slika 74). To datoteko vključimo v datoteko `Kconfig` višjega imenika.

60 Distribucija poleg jedra operacijskega sistema Linux vključuje še različne druge programe, ki omogočajo delo z računalnikom.

61 Napake, ki jih zagrešimo v jedru operacijskega sistema, pogosto povzročijo, da moramo računalnik ponovno zagnati.

62 Seveda gre za navidezno strojno opremo.

```
#
# 802.1d Ethernet Bridging
#

config UBRIDGE
    tristate "Universal Ethernet Bridging"
    ---help---
      If you say Y here, then your Linux box will include
      Universal Bridge.
      This framework by itself provides only the most modest
      of bridge functionalities -- essentially it mimics
      plain Ethernet switch.

      To compile this code as a module, choose M here: the
      module will be called ubridge.

      If unsure, say N.

config UBRIDGE_LOCAL
    bool "Local interface"
    depends on UBRIDGE
    default y
    ---help---
      If you say Y here, then the Universal Bridge will
      include a special loopback Ethernet interface that
      you can attach a local IPv4 or Ipv6 address to. You
      will be able to create multiple instances of these
      local interfaces and attach them all to Universal
      Bridge.

      Say N to exclude this support and reduce the binary
      size.

      If unsure, say Y.

...
```

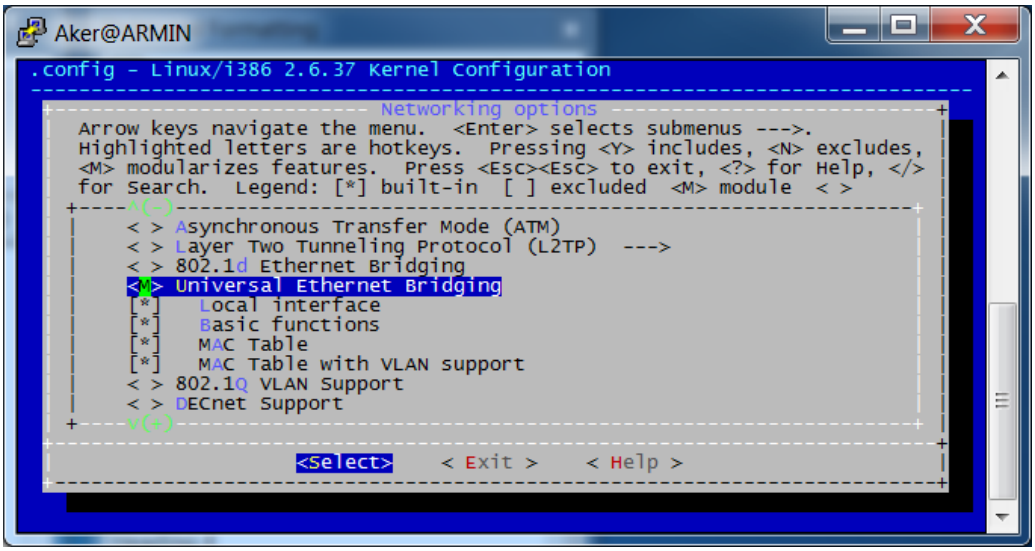
Slika 74: Datoteka `net/ubridge/Kconfig`.

2. Pripravimo opis navodil za prevajanje v datoteki `Makefile` (slika 75). To datoteko vključimo v datoteko `Makefile` višjega imenika.

```
#
# Makefile for Universal Bridging.
#

EXTRA_CFLAGS += -DDEBUG

obj-$(CONFIG_UBRIDGE)              += ubridge.o

ubridge-y                          += ubr.o
ubridge-y                          += ubr_framework.o
ubridge-y                          += ubr_if.o
ubridge-$(CONFIG_UBRIDGE_LOCAL)    += ubr_local.o
ubridge-$(CONFIG_PROC_FS)          += ubr_proc.o
ubridge-y                          += ubr_process.o
ubridge-y                          += ubr_utils.o
ubridge-$(CONFIG_UBRIDGE_BASIC)    += plugin/ubr_basic.o
ubridge-$(CONFIG_UBRIDGE_MT)       += plugin/ubr_mt.o
ubridge-$(CONFIG_UBRIDGE_MT_VLAN)  += plugin/ubr_mt.o
```

Slika 75: Datoteka `net/ubridge/Makefile`.

Rezultat je viden na sliki 76.



Slika 76: Vključitev univerzalnega mosta v prevajanje jedra Linux.

11.3.2 Vključitev prilagodljivega modularnega mosta v operacijski sistem

Prilagodljivi modularni most se v operacijski sistem vključi kot naložljivi modul (angl. *loadable module*). Taka oblika je še posebej primerna v času razvoja, saj omogoča, da module iz jedra odstranimo, popravimo in ponovno naložimo. Na ta način se izognemo ponovnemu zagonu celotnega operacijskega sistema, to pa prihrani veliko časa.

Ob zagonu univerzalni most ustvari datoteko `/proc/net/ubr/ubr`, preko katere lahko v nadaljevanju spremljamo in nadzorujemo dogajanje v univerzalnem mostu. Za nadzor

```
cat rules.txt > /proc/net/ubr/ubr

cat > /proc/net/ubr/ubr <<EOF
rule add dsl10 ingress 2 action FrameSet 15 0x1f 0x60
EOF

cat > /proc/net/ubr/ubr
```

Slika 77: Trije načini upravljanja mosta iz ukazne vrstice.

uporabljamo kar ukaze, ki smo jih v poglavju 10.3 navajali kot primere. Slika 77 prikazuje preprostost takega pristopa.

Za vsak primerek univerzalnega mosta nastane nova datoteka z imenom, ki je enako mostu. Za most `br0` tako nastane datoteka `/proc/net/ubr/br0`. Preko nje lahko spremljamo in nadzorujemo delovanje posameznega mosta.

Za vsak primerek mosta ustvarimo tudi novo omrežno napravo z njegovim imenom. Omrežno napravo lahko potem nadzorujemo s pomočjo sistemskega programa `ifconfig`.

11.4 Primerjava z ostalimi mostovi

Naš univerzalni most bomo v tem poglavju primerjali z mostom operacijskega sistema Linux, visoko integriranim in nizko integriranim mostom. Primerjav med njimi, ki smo jih opravili že v poglavju 8.7 , ne bomo ponavljali.

11.4.1 Spremembe

Izvajanje raznih sprememb je zelo enostavno – to je bil eden naših osnovnih ciljev. Pod določenimi pogoji je mogoče funkcionalnost spreminjati celo med delovanjem mosta. Pri tem nimamo v mislih sprememb v obliki nastavitev, ampak dodajanje in odvzemanje funkcionalnosti.

Dodajanje modulov, ki mostu priskrbijo dodatne pogoje in akcije, je mogoče med delovanjem in ne zahteva ponovnega zagona. To je predvsem pri mostu, ki je v stalni uporabi, zelo pomembno. Čas ponovnega zagona namreč neposredno občutijo uporabniki.

Med delovanjem je pod določenimi pogoji mogoče tudi varno odstraniti zunanje module. Če pogoji in akcije, ki jih zunanji modul prinaša, trenutno niso v uporabi, potem lahko celoten modul brez nevarnosti odstranimo.

Velik napredek v primerjavi z ostalimi tremi mostovi smo dosegli s tem, da smo most zasnovali kot modul, ki je na operacijski sistem vezan le toliko, kot je zares nujno potrebno (tak je tudi most operacijskega sistema Linux). Dodaten korak naprej pa pomeni še delitev mosta na osnovni infrastrukturni del in funkcionalni del. S pripravo prilagodljive infrastrukture smo olajšali spremembe funkcionalnosti.

11.4.2 Odkrivanje napak

Ena večjih odlik predstavljenega mosta je odlična podpora za odkrivanje napak. Pri tem

imamo v mislih predvsem količino informacij, ki so nam na voljo. Nivo informacij, gre za razmeroma surove podatke, zahteva od raziskovalca dobro poznavanje delovanja mostov in protokolov.

11.4.2.1 Napake v nastavitvah

Napake v nastavitvah z lahkoto najdemo – ob pogoju, da razumemo delovanje mosta. Pri tem nimamo v mislih konkretne izvedbe mosta, ampak poznavanje splošnega delovanja mostov in operacij, ki jih most izvaja. Na primer, če razumemo, kaj naj most naredi, ko sprejme določeno vrsto okvirja, potem iz opisov dogajanja v mostu lahko razberemo, če je obnašanje ustrezno.

Glede odkrivanja napak v nastavitvah torej lahko rečemo, da most ponuja odlična orodja, vendar pa njihova uporaba zahteva veliko znanja.

11.4.2.2 Napake v izvedbi

Pri odkrivanju napak v izvedbi si lahko pomagamo s sledjo, ki jo okvir „*pušča za sabo*“ med obdelavo v mostu. Za odkrivanje nekaterih napak je to dovolj.

Pri odkrivanju drugih napak se moramo zanašati na informacije, ki jih pogoji in akcije izpisujejo, kadar od njih to zahtevamo. Količina in vrsta informacij, ki jih na ta način lahko pridobimo, je odvisna izključno od izvedbe posameznega pogoja ali akcije.

Kot skrajno možnost lahko uporabimo izpisovanje okvirja. V toku obdelave lahko to naredimo večkrat, zavedati pa se moramo, da tovrstno izpisovanje lahko močno upočasni delovanje.

11.4.2.3 Sled okvirja

Vsi trije mostovi, ki jih uporabljamo za primerjavo, so glede odkrivanja napak zelo skopi. Očitno je, da so bili avtorji v vseh primerih prepričani ne le v pravilnost delovanja, ampak tudi v pravilnost nastavitev. Iz izkušenj lahko rečemo, da sta obe predpostavki pogosto napačni.

Sled okvirja, tako rečemo opisu korakov, ki jih na poti skozi most opravi okvir, zahteva visoko stopnjo poznavanja delovanja mosta, zato je kot pripomoček zahtevno orodje. Ponuja pa prav vse, kar za odkrivanje napak potrebujemo:

- prikaz začetnega stanja (vhodni vmesnik, prejeti okvir);
- izpis posameznih izvajalnih korakov (uspešnost preverjanja posameznih pogojev, izvedene akcije) in
- prikaz končnega stanja (odmetavanje okvirja ali njegovo pošiljanje preko enega ali več vmesnikov).

Ogrodje mosta ne more vsiliti standarda izpisovanja sledi tudi pogojem in akcijam, ki so v most vključeni od zunaj, pa njihove avtorje preko ponujenih vmesnikov in z zgledom k temu vseeno spodbuja.

Pogosto so razlogi za pomanjkljivo podporo sledenju v zmogljivosti sistema. Zaradi tega smo se posebej potrudili, da je podpora za sledenje pri običajnem delu kar se da neopazna. Največji korak v to smer predstavlja način vključitve te podpore in njeno trajanje.

Vključitev in izključitev podpore za sledenje lahko dosežemo preko upravljalnega vmesnika

ali preko akcije. Prvo je uporabno, ker nam omogoča prikaz zares celotne sledi, drugo pa je koristno, ko nas zanima sled le za določeno vrsto prometa, saj lahko s pogoji določimo vrsto prometa, ki mu bomo sledili.

Trajanje prikaza sledi je lahko neomejeno, lahko pa sledimo le določenemu številu okvirjev. Predvsem slednje je omogočilo, da smo lahko sledenje uporabili tudi med običajnim delovanjem in ne le v laboratorijskih pogojih[63].

11.4.3 Podvajanje

Trenutna izvedba nima praktično nikakršnega podvajanja funkcionalnosti. Vse je zasnovano okrog kar se da velike splošnosti in ponovne uporabljivosti.

11.4.3.1 Razlogi za podvajanje

Ob dodajanju zunanjih modulov lahko pričakujemo, da bo vseeno prišlo do določenih podvajanj. Tipičen primer so priročne funkcije, ki nam olajšajo delo, pa niso del splošno uporabljanih knjižnic. Kor primer lahko navedemo funkcijo, ki pretvori numerično shranjen naslov IPv4 v niz. Tako funkcijo pogosto potrebujemo, a ni del nobene sistemske knjižnice. Zaradi tega obstaja več funkcionalno enakih izvedb, ki pa se v podrobnostih lahko razlikujejo.

11.4.3.2 Splošnost proti podvajanju

Največji korak proti podvajanju smo dosegli s prehodom na manjše število splošnih pogojev in akcij.

Odličen primer je delo z značko VLAN, kjer imamo opravka s tremi deli (prioriteta, DEI/CFI in VID), prej pa moramo vedno preveriti, da v resnici delamo z značko VLAN. Včasih je bilo zadosti, da smo preverili ali je v okvirju vsebovan promet označen z Ethernetovim tipom `0x8100`, danes je poleg tega treba podpreti še tip `0x88a8`, nekaj časa je bil v uporabi tudi tip `0x9100` ... V vseh primerih pa sta oblika in pomen preostanka značke enaka. Če nadomestimo preverjanje točno določenega tipa s preverjanjem nastavljive vrednosti, potem smo že dosegli večjo splošnost.

Drugi primer splošnosti, ki prinaša še večje prihranke, pa je izdatna uporaba pristopa maska-vrednost. Na ta način lahko z enim pogojem ali akcijo, podpremo veliko število pogojev in akcij, ki bi bili prilagojeni posameznim ožjim uporabam.

11.4.4 Upravljanje

Upravljanje univerzalnega mosta je zelo zahtevno. Splošen pregled razlik v primerjavi z visoko in nizko integriranim mostom pokaže, da smo veliko sprememb dosegli predvsem z odstranitvijo vsakršne navezave na nivo upravljanja. Univerzalni most se zelo malo zaveda pomena podatkov, ki jih obdeluje.

Upravljavec univerzalnega mosta mora dobro poznati operacije, ki jih most izvaja nad okvirji Ethernet. Brez tega je pravilna nastavitev nemogoča. Zaradi zahtevnosti upravljanja je tem bolj pomembno, da je odkrivanje napak razmeroma preprosto.

63 Sledenje je procesorsko zahtevno, zato prikaz velike količine sledi negativno vpliva na prepustnost celotnega mosta. Če sledenje vključimo na terenu, pri tem pa to ne podpira samodejne izključitve, potem tvegamo, da bo most tako obremenjen, da sledenja ne bomo več uspeli izključiti, preden bodo za to poskrbeli drugi varnostni mehanizmi (čuvaj).

Predvsem visoko in nizko integrirani most pri izvedbi v veliki meri sledita upravljavskemu pogledu na omrežje. To zelo poenostavlja upravljanje, saj se v večini primerov višji koncepti upravljanja neposredno preslikajo v izvedbene funkcije. Izvedba upravljanja je zato zelo enostavna.

Za razliko od tega pa naš most pristopa k obdelavi okvirjev z drugega zornega kota. Akcije in pogoje smo zasnovali v prvi vrsti z mislijo na možno. Pri preverjanju in spreminjanju okvirjev smo torej ponudili uporabniku vse, kar je mogoče, priročnost pa ni bila naš prvi namen. Rezultat je manjše število univerzalnih komponent, katerih uporaba pa je zahtevnejša.

Čeprav lahko zapletenost upravljanja omenimo kot korak nazaj v primerjavi z visoko in nizko integriranim mostom (most operacijskega sistema Linux je navkljub svoji enostavnosti za upravljanje primerljivo zahteven), pa ovijanje v dodatni nivo, ki bi poenostavil upravljanje, ne bi bilo zahtevno. Težava je le v tem, da bi s takim ovojem pravzaprav izničili velik del prednosti, ki smo jih z drugačno arhitekturo dosegli.

11.4.5 Hitrost

Izvedba tako odprtega mosta je težko tako zelo optimizirana, kot to velja za bolj zaprte mostove, vseeno pa je pomembno poudariti, da tukaj presenečenja niso redka. Ne smemo namreč pozabiti, da je večina algoritmov, ki optimizirajo katero od pogosto izvajanih opravil, nastala, ko so bili običajni enostavni pomnilniki. Današnji pomnilniški modeli od tega močno odstopajo.

Če danes želimo hiter algoritem, potem v prvi vrsti moramo preprečiti zgrešitve v predpomnilniku (angl. *cache miss*), šele potem so pomembni drugi pogledi. Tega pomembnega dejstva se mnogi starejši algoritmi ne zavedajo. Rezultat je lahko vse prej kot dober [55].

Tega ne omenjamo v bran hitrosti univerzalnega mosta, pač pa kot razlago za včasih nepričakovane rezultate. Naš most se v preprostih postavitvah obnese zelo dobro, celo najbolje.

Primerjava z mostom operacijskega sistema Linux je zaradi njegove omejenosti mogoča le pri najbolj osnovnih testih (izključno premoščanje).

Primerjava z visoko in nizko integriranim mostom pokaže, da je naš most pogosto hitrejši. Razlog je v načinu izvedbe ostalih dveh. Pri njiju so namreč posamezni deli (delo z VLAN označitvami, filtriranje, zagotavljanje kvalitete storitev, aplikacijski filtri) izvedeni kot ločeni gradniki, okvir pa prehaja od enega do drugega. Okvir vsakič opravi celotno pot, tudi v primerih, ko posamezni sklop z njim ne naredi nič. V takih primerih se naš most obnese bolje.

Slabše pa jo odnese v primerih, ko je funkcionalnost drugih dveh mostov polno izkoriščena. V takih primerih je naš most običajno malenkost počasnejši.

11.4.6 Robustnost

Robustnost je slaba točka predstavljenega mosta. Izveden je v jedru operacijskega sistema, zato je večina napak usodnih. Na srečo pa tudi ta težava ni nerešljiva.

Omenjena težava je pravzaprav težava operacijskih sistemov, ki so zgrajeni okrog velikega jedra. Obstaja tudi drugačen pristop, pri katerem je jedro majhno in vanj praktično ne

posegamo, vse ostalo dogajanje pa poteka v varnejšem uporabniškem načinu [21]. Na žalost to prinaša slabšo zmogljivost [20], a rast hitrosti strojne opreme tudi ta problem zmanjšuje.

Brez posebnih operacijskih sistemov pa bi bilo mogoče most funkcionalno enakovredno izvesti tudi v uporabniškem načinu. S tem bi bili problemi robustnosti do določene mere rešeni. Tak pristop ima še dodaten čar: delo z omrežnimi protokoli odpre širšemu krogu razvijalcev, ki se sicer dela v jedru izogibajo. Ob primerni zasnovi – v tem delu predstavljena arhitektura modularnega mosta to vsekakor je – je mogoče razvoj in preverjanje mosta v celoti opraviti v uporabniškem naslovnem prostoru, potem pa most z minimalnim naporom preseliti v jedro. Na ta način dobimo najboljše od obeh svetov.

11.4.7 Modularnost

Predstavljeni most je modularen v vseh pogledih. Jedro mosta je zelo majhno in vključuje le najbolj osnovne funkcije. Pravzaprav je posvečeno predvsem vključevanju zunanjih funkcionalnosti.

Osnovne funkcije so v most dodane ob uporabi prej omenjenih splošnih funkcij. To seveda pomeni, da je tudi osnovne funkcije zelo enostavno mogoče zamenjati z drugačnimi izvedbami.

Jedro mosta je izvedeno v obliki dinamično naložljivega modula. Brez potrebe po ponovnem zagonu ga lahko naložimo, odstranimo, zamenjamo za drugo različico itd.

Jedro mosta in operacijskega sistema Linux vsebujeta potrebne nastavke, da lahko tudi funkcionalne razširitve vključimo v obliki zunanjih modulov.

Kot poseben prikaz modularnosti smo večji del funkcionalnosti mosta, vključno z zelo osnovnimi pogoji in akcijami, v most vključili preko odprtih vmesnikov, ki so na voljo tudi vsem ostalim razvijalcem.

12 Zaključek

Mostovi protokola Ethernet obstajajo že dolgo, uporaba v vedno bolj zapletenih omrežjih pa narekuje razvoj novih funkcionalnosti zanje. Osnovno nalogo, preklapljanje okvirjev, danes dopolnjujejo različne napredne funkcije, ki omogočajo boljšo izrabo omrežja, večjo varnost in boljši nadzor nad delovanjem omrežja.

Ogledali smo si nabor funkcionalnosti, ki jih ponuja družina omrežnih mostov DSLAM. Gre za hitro razvijajoče mostove, kjer je potreba po dodajanju in spreminjanju funkcionalnosti zelo izrazita. Ogledali smo si, kako različne rešitve omogočajo odkrivanje napak – v nastavitvah in izvedbi funkcionalnosti.

Na podlagi primerjave treh različni izvedb smo spoznali, da razvijalci ubirajo različne pristope: visoko stopnjo integracije in prilagoditve določeni strojni opremi; zelo splošen pristop, povsem neodvisen od strojne opreme; vmesne poti. Posledice njihovih izbir se odražajo v načinu uporabe, zmogljivostih, funkcionalnosti in enostavnosti vzdrževanja nastalih mostov. Izpostavili smo zaželene lastnosti in poudarili mostove, ki te lastnosti imajo, pri ostalih pa navedli razloge za odsotnost teh lastnosti.

Razvili smo koncept modularnega Ethernetovega mosta, ki ga lahko v uporabimo kot nadomestek za obstoječe DSLAM mostove. Razvili smo tudi preprosto razširljiv način nastavljanja takih mostov in prikazali, kako z uporabo preprostih osnovnih operacij, ki jih dodamo modularnemu mostu, izvedemo bolj zapletene operacije, ki so običajne pri uporabi DSLAM mostov.

Koncept modularnega mosta smo podkrepili s praktično izvedbo za operacijski sistem Linux. Most je razvit v obliki modula, ki ga lahko prevedemo neposredno v jedro Linuxa, ali pa ga prevedemo kot modul, ki ga lahko med delovanjem dodamo ali odstranimo iz jedra.

Principu modularnosti sledijo tudi pogoji in akcije, ki razširjajo osnovno funkcionalnost mosta. Tudi ti so pripravljeni v obliki modulov, ki so lahko prevedeni v jedro Linuxa, lahko so pridruženi osnovnemu modulu mosta, lahko pa se z njim povezujejo kot ločeni moduli, ki jih dodajamo in odvzemamo med delovanjem.

Modularni most je zgrajen okrog majhnega števila zelo splošnih operacij, uporabniku pa prepušča njihovo pravilno uporabo. Slednje zahteva dobro poznavanje delovanja mosta, kar zna biti v nekaterih primerih težavno. To neprijetnost lahko odpravimo tako, da med most in uporabnika vrinemo nivo, ki uporabniku razumljive operacije prevede v take, ki jih razume most.

Po zmogljivostih modularni most sicer najboljših ne dosega, a to ni bil naš glavni namen. Zmogljivosti bi se kazalo posvetiti v prihodnosti.

13 Literatura

[1] Robert M. Metcalfe, David R. Boggs, *„Ethernet: distributed packet switching for local computer networks“*, Communications of the ACM, št. 7, zv. 19, str. 395–404, 1976.

[2] Hubert Zimmermann (1980): *OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection*, IEEE Transactions on Communications, zv. 28, št. 4, str. 425–432.

[3] Andrew S. Tanenbaum (1996): *Computer Networks, 3rd Edition*, ISBN 0-13-349945-6, Prentice-Hall.

[4] Radia Perlman (1992): *Interconnections: Bridges and Routers*, ISBN 0-201- 56332-0, Addison-Wesley.

[5] *Media Access Control (MAC) Bridges*, IEEE Standard 802.1D-2004.

[6] *Virtual Bridged Local Area Networks*, IEEE Standard 802.1Q-2005.

[7] *Virtual Bridged Local Area Networks, Ammendment 4 – Provider Bridges*, IEEE Standard 802.1ad-2005.

[8] *Virtual Bridged Local Area Networks, Ammendment 5: Connectivity Fault Management*, IEEE Standard Draft, 802.1ag/D8.1, 18.6.2007.

[9] A. Cohen, E. Shrum, D. Allan, D. Thorne, *“Migration to Ethernet-based DSL Aggregation”*, Broadband Forum, TR-101, april 2006.

[10] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan (2002): *RFC 3376 – Internet Group Management Protocol, Version 3*, IETF.

[11] David C. Plummer (1982): *RFC 826 – An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses*, IETF.

[12] Information Sciences Institute, University of Southern California (1981): *RFC 791 – DARPA Internet Program, Protocol Specification*, IETF.

[13] S. Deering, R. Hinden (1998): *RFC 2460 – Internet Protocol, Version 6 (IPv6)*, IETF.

[14] *Media Access Control (MAC) Bridges,* IEEE Standard 802.3ah-2004.

[15] R. Bush, D. Meyer (2002): *RFC 3439 – Some Internet Architectural Guidelines and Philosophy*, IETF.

[16] W. D. Sincoskie, C. J. Cotton (1988): *Extended bridge algorithms for large* networks, IEEE Network, zv. 2, št. 1, str. 16–24.

[17] Thomas L. Rodeheffer, Chandramohan A. Thekkath, Darell C. Anderson (2000): *SmartBridge: a scalable bridge architecture*, ACM SIGCOMM Computer Communication Review, zv. 30, št. 4, str. 205–216.

[18] Stephen E. Deering, David R. Cheriton, *“Multicast Routing in Datagram Internetworks and Extended LANs”*, ACM Transactions on Computer Systems, št. 2, zv. 8, str. 85–110, 1990.

[19] Rinat Khoussainov, Ahmed Patel, *„LAN security: problems and solutions for Ethernet networks“*, Journal Computer Standards & Interfaces, št. 3, zv. 22, str. 191–202, 2003.

[20] Härtig, Hermann; Hohmuth, Michael; Liedtke, Jochen; Schönberg, Sebastian, *„The performance of µ-kernel-based systems"*, Proceedings of the sixteenth ACM symposium on Operating systems principles, št. 5, zv. 31, str. 66–77, 1997.

[21] Andrew S. Tanenbaum, Jorrit N. Herder, Herbert Bos, *„Can We Make Operating Systems Reliable and Secure"*, Computer, zv. 39, str. 44–51, 2006.

[22] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, Gary J. Minden, *„A Survey of Active Network Research"*, IEEE Communications Magazine, zv. 35, str. 80–86, 1997.

[23] David L. Tennenhouse, David J. Wetherall, *„Towards an Active Network Architecture"*, Computer Communication Review, zv. 26, str. 5–18, 1996.

[24] Norman C. Hutchinson, Larry L. Peterson, *„The x-Kernel: An Architecture for Implementing Network Protocols"*, IEEE Transactions on Software Engineering, zv. 17, str. 64–76, 1991.

[25] Mathew J. Castelli (2004): *LAN Switching First-Step*, Cisco Press.

[26] Werner Bux, *„Performance issues in local area networks"*, IBM Systems Journal, št. 4, zv. 23, 1984.

[27] David R. Boggs, Jeffrey C. Mogul, Christopher A. Kent, *"Measured Capacity of an ethernet: Myths and Reality"*, ACM SIGCOMM Computer Communication Review, št. 2, zv. 19, str. 1–10, 1988.

[28] Werner Bux, *"Local area subnetworks: A performance comparison"*, IEEE Transactions on Communications, št. 10, zv. 29, str. 1465–1473, 1981.

[29] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *„RFC 2475 – An Architecture for Differentiated Services"*, IETF, 2009.

[30] Guillermo Mario Marro (2003): *Attacks at the Data Link Layer*, University of California, magistrsko delo.

[31] Xin Sun, Yu-wei E. Sung, Sunil D. Krothapalli, Sanjay G. Rao (2009): *A Systematic Approach for Evolving VLAN Designs*.

[32] Paul Kish (2000): *The Effect of Network Cabling on Bit Error Rate Performance*, NORDX/CDT.

[33] Jure Fritz (2007): *Zagotavljanje kakovosti storitev v Ethernet/IP/MPLS omrežjih, diplomsko delo*, Univerza v Ljubljani, Fakulteta za elektrotehniko.

13.1 Ostali viri

[34] *The Network Simulator – ns-2.* Dostopno na: http://www.isi.edu/nsnam/ns/

[35] George Varghese, *„Chapter 1 The Story of Bridging"*. Dostopno na: http://cseweb.ucsd.edu/users/varghese/TEACH/cs123/bridging.pdf

[36] *Cut-Through and Store-and-Forward Ethernet Switching for Low-Latency Environments*, Cisco Systems Inc., 2008. Dostopno na: http://www.ciscosystems.co.at/en/US/prod/collateral/switches/ps9441/ps9670/white_paper_c11-465436.pdf

[37] Dan DiNicolo: *Layer 2 Switching And Bridging*. Dostopno na: http://archive.networknewz.com/2004/0505.html

[38] Eric L. Michelsen: *Repeating, Bridging, Switching, Routing*, Inductive Logic. Dostopno na: http://www.google.com/url?sa=t&source=web&cd=10&ved=0CE4QFjAJ&url=http%3A%2F%2Fwww-physics.ucsd.edu%2F~emichels%2FWe%2520Deliver.ppt&ei=-TUWTLD9AtafOPKsjfgL&usg=AFQjCNEY_wB_ZxcemLYw1xO1LYa2uUCopg&sig2=MX-vPBnURfTrWMqOaySD8A

[39] *Switches & Bridges*. Dostopno na: http://www.firewall.cx/switches.php

[40] *Internetworking Technology Handbook, Chapter 4*. Dostopno na: http://www.cisco.com/en/US/docs/internetworking/technology/handbook/ito_doc.html

[41] *The Linux Kernel Archives*. Dostopno na: http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.37.3.tar.bz2

[42] M. Chris Riley, Ben Scott (2009): *Deep Packet Inspection: The End of Internet as We Know It?*, Freepress. Dostopno na: http://www.freepress.net/files/Deep_Packet_Inspection_The_End_of_the_Internet_As_We_Know_It.pdf

[43] IANA: *Ether Types*. Dostopno na: http://www.iana.org/assignments/ethernet-numbers

[44] IANA: *Protocol Numbers*. Dostopno na: http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml

[45] *CPU History*. Dostopno na: http://www-users.cs.york.ac.uk/~pcc/pc_history/index.html

[46] *Berkeley Software Distribution*. Dostopno na: http://www.bsd.org/

[47] The MINIX 3 Operating System. Dostopno na: http://www.minix3.org/

[48] Wind River: *Wind River VxWorks – Embedded RTOS with support for POSIX and SMP*. Dostopno na: http://www.windriver.com/products/vxworks/

[49] *Debian – The Universal Operating System*. Dostopno na: http://www.debian.org/

[50] VMware: *VMware Server, Free Virtualization Download for Virtual Server Consolidation*. Dostopno na: http://www.vmware.com/products/server/overview.html

[51] *VirtualBox*. Dostopno na: http://www.virtualbox.org/

[52] Eclipse Foundation Inc.: *Eclipse - The Eclipse Foundation open source community website*. Dostopno na: http://www.eclipse.org/

[53] FoxNews.com: *Internet Traffic from U.S. Government Websites Was Redirected Via Chinese Networks*. Dostopno na: http://www.foxnews.com/politics/2010/11/16/internet-traffic-reportedly-routed-chinese-servers/

[54] The Next Web: *China Hijacked 15% of US Internet Traffic-and no one noticed.*

Dostopno na: http://thenextweb.com/apps/2010/11/16/china-hijacked-15-of-us-internet-traffic-and-no-one-noticed/

[55] Poul-Henning Kamp: *You're Doing It Wrong*. Dostopno na: http://queue.acm.org/detail.cfm?id=1814327

[56] SpeedTest.net. Dostopno na: http://www.speedtest.net/

14 Izjava o samostojnosti dela

Izjavljam, da sem magistrsko nalogo izdelal samostojno pod mentorstvom prof. dr. Nikolaja Zimica. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Tomo Ceferin