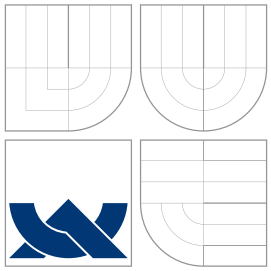
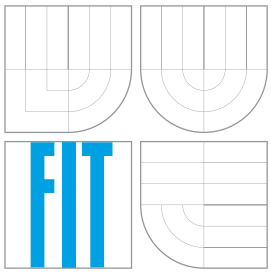


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NETGRAPH MODUL VE FREEBSD PRO POČÍTÁNÍ STATISTIK

NETGRAPH MODULE IN FREEBSD FOR TRAFFIC ACCOUNTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN BLAŽEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. RUDOLF ČEJKA
SUPERVISOR

BRNO 2007

Zadání bakalářské práce/1949/2006/xblaze17

Vysoké učení technické v Brně - Fakulta informačních technologií

Centrum výpočetní techniky Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Blažek Jan**
Obor: Informační technologie
Téma: **Netgraph modul ve FreeBSD pro počítání statistik**
Kategorie: Operační systémy

Pokyny:

1. Seznamte se s operačním systémem FreeBSD.
2. Seznamte se s síťovými moduly Netgraph ve FreeBSD.
3. Navrhněte Netgraph modul ve FreeBSD pro počítání přenosových statistik (účtování TCP/IP provozu).
4. Implementujte navržený modul.
5. Srovnejte tento modul s existujícími moduly.

Literatura:

- Zdrojový kód a dokumentace operačního systému FreeBSD (http://www.FreeBSD.org/)

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 a 2.

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

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

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

Vedoucí: **Čejka Rudolf, Ing.**, CVT FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

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

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Jan Blažek**
Id studenta: 84455
Bytem: V Mišpulkách 800, 284 01 Kutná Hora
Narozen: 15. 11. 1984, Kutná Hora
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Netgraph modul ve FreeBSD pro počítání statistik
Vedoucí/školitel VŠKP: Čejka Rudolf, Ing.
Ústav: Centrum výpočetní techniky
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	..... Jan Blažek .....
Nabyvatel	Autor

Abstrakt

Tato bakalářská práce se zabývá modulárním síťovým subsystémem Netgraph v jádře operačního systému FreeBSD. Netgraph se zde představuje z uživatelského hlediska. Je zde popsáno několik konkrétních modulů, nástroje pro práci s Netgraphem a příklady použití. Součástí práce je i implementace modulu pro sledování síťového provozu a počítání statistik. V této souvislosti jsou představeny i další subsystémy jádra využívané modulem – zaveditelné moduly jádra, mbuf – správa paměti pro meziprocesovou komunikaci v jádře a rozhraní sysctl pro sdílení proměnných mezi kernelem a userspace programy.

Klíčová slova

Netgraph, FreeBSD, síťové statistiky, TCP/IP, Ethernet, KLD, zaveditelné moduly jádra, sysctl, mbuf, BSD, UNIX

Abstract

This bachelor's thesis deals with modular kernel networking subsystem Netgraph in FreeBSD operating system. Netgraph is shown from user's point of view. This work describes several concrete modules, tools for Netgraph management and examples of usage. Part of the work is about implementation of Netgraph module for network traffic accounting. There are described some other kernel subsystems used by the module – kernel loadable modules, mbuf – memory management in kernel interprocess subsystem and sysctl interface for sharing kernel variables with user space programs.

Keywords

Netgraph, FreeBSD, network traffic accounting, TCP/IP Ethernet, KLD, kernel loadable module, sysctl, mbuf, BSD, UNIX

Citace

Jan Blažek: Netgraph modul ve FreeBSD pro počítání statistik, bakalářská práce, Brno, FIT VUT v Brně, 2007

Netgraph modul ve FreeBSD pro počítání statistik

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Rudolfa Čejky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Jan Blažek

13. května 2007

Poděkování

Děkuji Ing. Rudolfu Čejkovi za hodnotné rady a odborné vedení během mé práce.

Obsah

Kapitola 1

Úvod

Při správě počítačové sítě je důležité vědět, co se v síti děje, jaký typ dat a jaké množství dat teče přes naše servery. Když víme, co se v síti děje, můžeme dělat správná rozhodnutí. Tato práce se zabývá návrhem a implementací jaderného modulu, který může být použit pro sledování síťového toku. Modul má spíše jen základní funkčnost a pro reálné nasazení, by bylo vhodné ho dále rozšířit. Důraz v práci je kladen zejména na seznámení se s Netgraphem, s jehož pomocí je modul vytvořen. Dále jsou popsány i další oblasti, které s tímto modulem bezprostředně souvisí.

Kapitola 2 stručně pojednává o historii operačního systému FreeBSD a o jeho předchůdcích BSD a UNIXu. Je zde popsán vývojový model projektu FreeBSD a je zde přestavena hlavní licence operačních systémů BSD – BSD licence.

Kapitola 3 se zabývá samotným Netgraphem. Netgraph je představen z uživatelského hlediska, jsou zde vysvětleny základní pojmy a popsány některé moduly a nástroje pro práci s Netgraphem. Připraveny jsou i dva příklady využití Netgraphu.

V kapitole 4 jsou představeny další jaderné subsystémy využitelné pro modul na počítání statistik. Z trochu širšího pohledu jsou uvedeny moduly jádra. Dále je zde představena správa paměti pro meziprocesovou komunikaci v jádře a rozhraní sysctl pro sdílení proměnných mezi jádrem a userspace programy.

V předposlední kapitole 5 je popsán návrh a implementace modulu pro vytváření statistik. Jsou zde uvedena schémata hlaviček několika síťových protokolů a zde popsán princip jejich zkoumání modulem.

Závěrečná kapitola 6 srovnává modul s podobnými systémy a předkládá návrhy na rozšíření modulu.

Kapitola 2

Operační systém FreeBSD

Máme-li si povědět něco o historii operačního systému FreeBSD, musíme začít od jeho kořenů.

2.1 UNIX

Počátky UNIXu můžeme datovat do roku 1969. Tenkrát vznikl v hlavě Kenna Thompsona v AT&T Bell Laboratories nápad vytvořit nový operační systém. Záhy se k němu připojil Dennis Ritchie, autor jazyka C. Většina systému byla přepsána z čistého assembleru do jazyka C, což umožnilo jeho pozdější expanzi na spoustu různých platforem. Ritchie, Thompson a další vývojáři původně pracovali na vývoji operačního systému Multics, ze kterého byla přejata řada myšlenek a konceptů. Vývoj Multicsu byl velice rozsáhlý a nákladný projekt, který se zhroutil pod vlastní vahou. Vývojáři UNIXu se však nechtěli vzdát možností které jim Multics jako víceúlohový operační systém nabízel v době, kdy byly stále běžné systémy s dávkovým zpracováním úloh. Název UNIX je reakcí na Multics; v oblastech, kde se Multics snažil provádět mnoho úloh, UNIX měl provádět pouze jednu akci, ale zato dobře. Dalším předkem UNIXu byl CTSS (Compatilbe Time-Sharing System), předchůdce Multicsu vyvíjený na MIT v Bostonu. [6, 5]

2.2 BSD

Systém BSD je úzce spjat s Kalifornskou Univerzitou v Berkeley. Původní distribuce UNIXu ze 70. let obsahovaly zdrojové kódy systému a byly za příznivých podmínek dostupné hlavně pro univerzity.

Software z Berkeley byl vydáván v takzvaných Berkeley Software Distribucích (odtud zkratka BSD). Původně to nebyl celý operační systém, ale spíše rozšíření k UNIXu. První verze 1BSD (1977) byla doplňkem k šesté edici UNIXu a jejími hlavními komponenty byly překladač jazyka Pascal a řádkový editor ex.

Druhá verze 2BSD obsahovala aktualizace programů z 1BSD a dva nové, v UNIXu dodnes známé, programy; editor vi (vizuální rozhraní k editoru ex) a C shell.

Práce na verzi 3BSD byly ve znamení portování systému z architektury PDP-11 na VAX. Port na tuto architekturu sice již existoval, byl jím UNIX/32V od Bellových laboratoří, ale ten nevyužíval možností virtuální paměti VAXu. Jádro 32V bylo z velké části přepsáno studenty z Berkeley a tato práce si získala poměrně dobré uznání.

Úspěch 3BSD zapříčinil to, že grantová agentura ministerstva obrany USA DARPA (Defense Advanced Research Projects Agency) se rozhodla dotovat tým z Berkeley, aby tak podpořila jeho další vývoj. Cílem projektu 4BSD bylo vytvořit podporu pro internetový protokol TCP/IP, který vzešel právě od DARPY.

Následovaly verze 4.1, 4.2, 4.3. Ve verzi 4.2 byla v roce 1983 uvolněna implementace protokolu TCP/IP a byla tak výrazně posílena podpora sítí v systému UNIX, která nebyla do té doby nijak zvláště rozvinutá.

Do doby verze 4.3 obsahovaly distribuce BSD kód z AT&T UNIXu a vyžadovaly tak vlastnictví jeho licence. Tato licence však byla poměrně drahá a vznikala poptávka po rozšířeních BSD (hlavně síťovém kódu) bez těchto omezení. Reakcí bylo vydání Networking Release (Net/1) v roce 1989, které bylo uvolněno pod BSD licencí.

Po verzi Net/1 se vývojář Keith Bostic zamýšlel vydat větší část systému pod BSD licencí. Začal tedy znovu implementovat některé nástroje bez použití kódu pocházejícího z AT&T. Například editor vi, který obsahoval kód z licencovaného editoru ed, byl přepsán jako nvi. Po jednom a půl roce, byl přepsán skoro všechen kód od AT&T kromě několika zdrojových souborů kernelu. Ty byly vyjmuty a byla uvolněna distribuce Net/2 jako téměř kompletní operační systém.

Systém Net/2 byl základem ke dvěma portům na architekturu Intel 80386. Byly to nekomerční 386BSD od Williama Jolitze a proprietární BSD/386 (později přejmenován na BSD/OS) od Berkeley Software Design (BSDi). Systém 386BSD byl základem pro projekty NetBSD a FreeBSD. [11]

2.3 FreeBSD

FreeBSD projekt se vyvinul v roce 1993 z neoficiálních rozšíření k 386BSD. V roce 1994 byl ukončen soudní spor ohledně vlastnictví práv k částem Net/2 mezi Univerzitou v Berkeley a firmou Novell, která je koupila od AT&T. Práva byla přiznána společnosti Novell a uživatelům Net/2 bylo doporučeno přejít k 4.4BSD-Lite, což byla verze, ze které byl důsledně odstraněn kód od AT&T a chyběly v ní velké kusy kódu, které byly potřeba pro sestavení funkčního systému. Projekt FreeBSD musel tedy začít vyvíjet podstatnou část znovu z kódu 4.4BSD-Lite. [9, kap. 1.3]

2.4 Vývojový model a vedení projektu FreeBSD

Při snaze pochopit jak se projekt FreeBSD vyvíjí a jak vzniká si musíme uvědomit některé aspekty vývoje opensource softwaru. Oproti vývojářům proprietárních systémů jsou vývojáři FreeBSD často pouze dobrovolníci, nejsou (až na některé výjimky) za odvedenou práci placeni a často programují hlavně pro zábavu ve svém volném čase.

Kompletní zdrojové kódy systému včetně dokumentace, hlášení o chybách a archiv příspěvků na mailing-listu je veřejně dostupný přes systém správy verzí CVS. Práva k zápisu do těchto repozitářů má však pouze omezený okruh lidí, tzv. *commiters*. Vývojáři, kterých je v současnosti cca 3000-4000, musejí spolupracovat s některým z 300-400 commiterů, jestliže chtějí, aby se jimi navrhované změny promítly do systému. Stejně jako vývojáři se i commiters většinou specializují na některou část systému. Všechny rozsáhlé změny by měly být ověřeny více commitery, než budou zaneseny do zdrojového stromu. V hierarchii výš nad commitery je už jen tzv. *core team*. Členové této užší skupiny (v současnosti 9 lidí) jsou voleni každé dva roky. Jejich úkolem je přijímání závažnějších rozhodnutí, řešení sporů dvou nebo více commiterů a přijímání nových commiterů.

Vývoj probíhá paralelně ve dvou větvích: FreeBSD-STABLE a FreeBSD-CURRENT. Větev FreeBSD-STABLE je zamýšlena jako stabilní a promítají se do ní pouze změny, které opravují chyby a přidávají podporu pro nový hardware. Stabilní větev je čas od času vydávána jako „RELEASE“ a je uvolněna v podobě ISO obrazů k vypálení na CD. Aktivní vývoj a přidávání nových možností probíhá ve větvi CURRENT. Přibližně každé dva roky se větev CURRENT rozdvojí, aby se zformovala nová větev STABLE.

2.5 BSD licence

Ačkoliv jsou části systému licencovány pod různými licencemi, preferovanou licencí pro jádro a systémové nástroje je BSD licence. BSD licence samotná má svůj původ na univerzitě v Berkeley. Text této licence je uveden v příloze A.

BSD licence je velmi volnou licencí, oproti GNU GPL neobsahuje žádná omezení, kromě toho, že sířené odvozené dílo (ať už v binární nebo zdrojové formě) musí obsahovat informaci o autorech, o licenci a běžné zřeknutí se odpovědnosti za dílo. BSD licence umožňuje komerční využití včetně využití bez zveřejnění zdrojových kódů. Někdy bývá vtipně označována jako *copycenter*:

> Existuje *copyright*, který zakazuje šíření díla, *copyleft*, který naopak zakazuje omezování šíření díla a BSD licenci nazvěme *copycenter*, což znamená „Vezměte si to do copycentra a udělejte si kopií kolik chcete.“ [10]

Kapitola 3

Síťový subsystém Netgraph

3.1 Co je to Netgraph?

Netgraph je modulární síťový subsystém implementovaný přímo v jádře operačního systému FreeBSD. Úkolem Netgraphu není nahradit existující síťovou infrastrukturu v jádře, ale spíše se s ní doplňovat a nabídnout vývojářům nové možnosti [2]:

- flexibilní způsob kombinování protokolů a ovladačů síťových zařízení
- způsob jak modulárně implementovat nové protokoly
- společnou platformu pro vzájemnou komunikace objektů v kernelu
- efektivní implementaci v kernelu

Netgraph navrhli a poprvé implementovali vývojáři Archie Cobbs a Julian Elischer roku 1996 ve firmě Whistle Communications, Inc. ve verzi FreeBSD 2.2 modifikované pro zařízení Whistle InterJet. Jednalo se o server určený pro menší kanceláře a domácnosti k připojení do internetu a poskytování služeb jako e-mail, VPN, firewall, souborový server apod.[1] Do oficiálního stromu se Netgraph dostal poprvé ve verzi FreeBSD 3.4.

V návrhu Netgraphu lze poměrně snadno rozeznat ovlivnění unixovou filosofií. Místo nějakého složitého robustního systému je k dispozici celá sada několika relativně jednoduchých nástrojů, které provádějí jednoduché přesně definované operace. Díky tomu se tento systém dá jednoduše přizpůsobit a lze s ním provádět věci, na které původní návrháři třeba ani nepomysleli. Těmito nástroji jsou *uzly* (nodes), které jsou pomocí *hran* spojovány do větších celků, *grafů*. Podél hran se šíří data, která jsou zpracována v uzlech, např. přidáváním či odebíráním hlaviček paketům při implementaci různých protokolů.

3.1.1 Základní pojmy

V této podkapitole si objasníme základní pojmy jako uzly, hrany, kontrolní zprávy.

Uzel

Základní jednotkou v Netgraphu je uzel. Uzly zpracovávají data, která přes ně tečou, např. přidávají nebo odebírají hlavičky různých protokolů. Uzly také mohou být zdrojem nebo cílem dat v případě, že jsou asociovány přímo s hardware nebo s jinými částmi síťového subsystému. Všechny uzly mají několik společných předdefinovaných metod, které jim umožňují komunikovat s ostatními uzly.

Každý uzel patří k nějakému *typu*. Typ určuje chování uzlu a způsob připojení k ostatním uzlům. Nabízí se paralela s objektově orientovaným programováním. Typ můžeme chápat jako třídu, která je odvozená od hlavní třídy a dědí tak rysy společné pro všechny uzly. Uzel je instancí typu (třídy). Typ uzlu je popsán jednoznačných ASCII názvem. Uzlu je v době jeho vytvoření přidělena jednoznačná adresa, která je tvořena 32-bitovým číslem (*ID number*). Navíc může být uzel pro větší přehlednost a usnadnění práce pojmenován ASCII názvem. Název nesmí obsahovat znaky '.' nebo ':' a jeho délka je implementačně omezená na `NG_NODESIZ` (včetně ukončovacího znaku nula). Spojením páru háků vznikne hrana. Data tečou obousměrně mezi uzly podél hran tvořených párem háků. Uzel může mít libovolný počet háků.

Každý hák má svůj ASCII název, který je jednoznačný v rámci uzlu. Název nesmí podobně jako u názvu uzlu obsahovat znaky '.' a ':' a jeho délka je omezená na `NG_HOOKSIZ` znaků (včetně ukončovacího znaku nula). Hák je vždy připojen k nějakému jinému háku a vzniká atomicky až v době vytváření hrany. Odstranění háku na jedné straně spoje má za následek odstranění celé hrany i s protilehlým hákem. Pro háky můžou být definovány vlastní metody pro příjem dat a kontrolních zpráv. Dojde tak k „přetížení" obecných metod pro daný uzel.

Kontrolní zprávy

Ke komunikaci s uzly, k jejich konfiguraci a zjišťování stavu se používají kontrolní zprávy. Všechny uzly rozumějí základním kontrolním zprávám, které jsou součástí implementace Netgraphu. Tvůrce modulu může navrhnout vlastní zprávy specifické pro vytvářený uzel.

Zprávy jsou z důvodu efektivnosti v binárním formátu, ale většinou je navíc definován i textový formát, který je vhodnější pro ladění a použití v uživatelských rozhraních. Všechny zprávy jsou opatřeny 32-bitovou hodnotou nazývanou *typecookie*, která značí typ zprávy. Každý typ uzlu, který rozumí dané zprávě, má definovanou stejnou hodnotu typecookie.

3.1.2 Adresování

Netgraph nabízí přehledný způsob jak adresovat konkrétní uzel v grafu. Každý uzel je dostupný přes adresu, která má typ řetězce ASCII znaků. Je nutné poznamenat, že toto

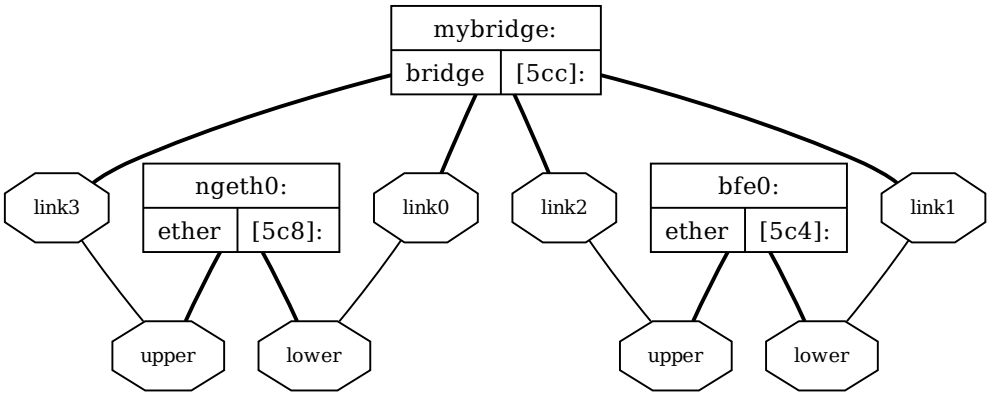
adresování je použitelné pouze pro zasílání kontrolních zpráv.

Všechny uzly v grafu mají unikátní ID (ID number). Uzavřením tohoto čísla v hexadecimálním tvaru do hranatých závorek a přidáním dvojtečky na konec dostaneme platnou adresu konkrétního uzlu. Například uzel s ID číslem 38f můžeme adresovat jako ,,`[38f]:`".

Některé uzly mají jména. Uzly spojené s hardwarovým zařízením mají v netgraphu typicky jméno daného zařízení. Například uzel asociovaný s ethernetovým adaptérem ,,`em0`" má adresu ,,`em0:`".

Adresy mohou být i složitější než pouhé jméno nebo ID číslo uzlu, navíc se adresy dělí na absolutní a relativní. Absolutní adresa začíná jménem nebo ID číslem uzlu, dále následuje dvojtečka, za ní posloupnost háků oddělená tečkami. Relativní adresa obsahuje pouze seznam háků oddělených tečkami. Cesta pak začíná v uzlu, ze kterého se odkazujeme. Pro názornost si uvedeme několik příkladů. Uvažujme obrázek 3.1, na uzel `ngeth0` se můžeme odkázat například těmito adresami:

```
ngeth0:
mybridge:link3
bfe0:lower.link0
[5c8]:
[5c4]:upper.link3
mybridge:link2.lower.link3
```



Obrázek 3.1: Příklad k adresování

3.2 Příklady modulů

Nyní si uvedeme několik příkladů konkrétních netgraphových modulů (resp. typů), které jsou implementovány ve FreeBSD. U každého popíšeme pouze základní chování a možnosti využití. Podrobnější informace lze nalézt v manuálových stránkách k příslušným modulům.

3.2.1 echo

Modul `ng_echo(4)` posílá zpět odesílateli všechny kontrolní zprávy a data, které k němu dorazily. Tento modul je použitelný zejména k testování a ladění.

3.2.2 tee

Modul `ng_tee(4)` funguje podobně jako příkaz `tee(1)`. Chová se jako rozdvojka, ale na rozdíl od jejího shellového ekvivalentu je oboustranná. Tee uzly mají čtyři háky: left, right, left2right a right2left. Všechna data, která dorazí na hák left, jsou poslána na háky right a left2right. Obdobně z right na left a right2left. Dále pak data z uzlu right2left (resp. left2right) tečou v nezměněné podobě na háky right (resp. left). Uzel typu tee se hodí zejména k ladění nebo „odposlouchávání“ spojení mezi dvěma uzly. Za zmínku stojí i to, že uzel si pro jednotlivé háky vytváří statistiku o počtu datových zpráv (paketů) a jejich celkové velikosti pro směr toku ven i dovnitř uzlu.

3.2.3 iface

Vytvořením uzlu typu iface vznikne virtuální síťové rozhraní typu point-to-point. To je dosažitelné jednak z Netgraphu jako uzel, ale i jako klasické síťové rozhraní přes systémový nástroj `ifconfig(8)`. Rozhraní jsou pojmenována jako ng0, ng1, atd. Uzly tohoto typu mají háky odpovídající jednotlivým protokolům (inet, inet6, ipx, atalk, atm, ...). Pakety, které dorazí na síťové rozhraní se objeví na háku příslušného protokolu a data poslaná na hák značící určitý protokol pošle rozhraní do sítě jako pakety daného protokolu.

3.2.4 eiface

Podobně jako `ng_iface(4)` i modul `ng_eiface(4)` vytváří virtuální síťové rozhraní s tím rozdílem, že se jedná o ethernetové rozhraní. Rozhraní jsou pojmenována názvy ngeth0, ngeth1, atd.

3.2.5 ether

Modul `ng_ether(4)` umožní síťovým ethernetovým rozhraním objevit se v Netgraphu jako uzly, které jsou pojmenovány stejně jako jim příslušející ethernetová rozhraní. Uzly typu ether mají háky *lower*, *upper* a *orphans*. Hák *lower* slouží k přímému spojení s ethernetovým rozhraním. Když je připojen, jsou na něj doručeny všechny příchozí pakety ze sítě. Zápis dat

na tento hák způsobí jejich odeslání do sítě. Hák *upper* vytváří spojení s horními vrstvami síťového stacku. Po jeho připojení jsou na něj posílány všechny odchozí pakety, místo aby byly vysílány přímo síťovým adaptérem. Zápis na tento hák způsobí, že data budou přijata jako ethernetový rámec jádrem OS, tak jako by se jednalo o data ze sítě. Hák *orphans* se chová podobně jako *lower* s tím rozdílem, že na jeho výstupu se objeví pouze pakety, u kterých nebyl rozpoznán jejich typ (tj. jádro OS si s nimi neumí poradit). Když nejsou žádné háky připojeny, uzel se chová tak, jako by byly háky *lower* a *upper* spojeny.

3.2.6 socket

Modul `ng_socket(4)` implementuje uzly, které jsou zároveň uzly netgraphu a zároveň párem soketů z rodiny `PF_NETGRAPH`. Jeden soket je použit pro data a druhý pro kontrolní zprávy. Uzel netgraphu je vytvořen v době vytváření soketů v userspace programu. Příklad takového vytvoření soketů:

```
int s1, s2;
s1 = socket(PF_NETGRAPH, SOCK_DGRAM, NG_CONTROL);
s1 = socket(PF_NETGRAPH, SOCK_DGRAM, NG_DATA);
```

3.2.7 ksocket

Modul `ng_ksocket(4)` vytváří uzly, které jsou zároveň BSD sokety a zároveň uzly netgraphu. Avšak v tomto případě jde o obrácenou verzi `ng_socket`. Tento modul nám umožní vytvořit BSD soket přímo v kernelu a navázat spojení s procesem, který běží lokálně na stejném stroji, nebo s jiným strojem přes síť. Uzel akceptuje kontrolní zprávy *connect, accept, listen, bind* apod., které mají stejný význam jako při použití `socket(2)`.

3.3 Nástroje pro práci s Netgraphem

V této sekci si popíšeme programové nástroje, které slouží k vytváření grafu a ke komunikaci s uzly. Dále si předvedeme některé techniky, které sice nejsou nezbytně nutné, ale výrazně nám zjednoduší práci.

3.3.1 nghook

Nástroj `nghook(8)` se připojí na nepřipojený hák uzlu v grafu a umožní přijímat a vysílat pakety přes standardní vstup a výstup. Výstup může být dekódován do čitelného ASCII formátu. Například pokud si chceme nechat zobrazit v ASCII formátu nerozpoznané pakety ze síťového adaptéru `bfe0`, použijeme příkaz:

```
nghook -a bfe0: orphans
```

K propojení netgraphu, který běží v kernelu, s procesem nghook v uživatelském prostoru je použit uzel typu socket (`ng_socket(4)`).

3.3.2 ngctl

Program `ngctl(8)` vytvoří uzel typu socket. Přes něj jsou uzlům zasílány kontrolní zprávy a získávány informace o uzlech. Tento program s rozhraním příkazové řádky může být jednak použit v interaktivním módu, ale také v shellových skriptech například pro dávkové vytvoření již navrženého schématu grafu. Výpis nejdůležitějších příkazů:

connect	Spojí pár háků, aby se vytvořila hrana.
dot	Vypíše celý graf ve formátu GraphViz (.dot).
help	Zobrazí seznam příkazů nebo podrobnou nápovědu pro specifický příkaz.
list	Zobrazí informace o všech uzlech v grafu.
mkpeer	Vytvoří nový uzel a připojí ho na existující uzel.
msg	Zašle kontrolní zprávu vybranému uzlu.
name	Přiřadí uzlu jméno.
rmhook	Rozpojí dva uzly, které jsou spojené.
show	Zobrazí informace o daném uzlu.
shutdown	Přeruší všechny hrany a odstraní uzel.
types	Zobrazí informace o právě nainstalovaných typech.
quit	Ukončí program.

Příkaz mkpeer má na rozdíl od ostatních příkazů složitější syntaxi, která nemusí být na první pohled zřejmá.

```
použití: mkpeer [cesta]: <typ> <hák1> <hák2>
```

Příkaz vytvoří nový uzel typu „typ“ a připojí ho na uzel na adrese „cesta“. Háky určené ke spojení jsou „hák1“ na původním uzlu a „hák2“ na nově vytvořeném uzlu.

Za povšimnutí stojí příkaz dot, který produkuje výstup pro sadu programů GraphViz. Po nainstalování této sady z binárního balíčku příkazem:

```
# pkg_add -r graphviz
```

případně kompilací a instalací z portů:

```
# cd /usr/ports/graphics/graphviz
# make install clean
```

nám následující příkaz zobrazí schéma grafu v grafické podobě.

```
# ngctl dot | dotty -
```

V balíku graphviz je více užitečných programů pro práci s tímto formátem. Za zmínku stojí alespoň `dot`, který umí mimo jiné export do PostScriptu. Takto byly vytvořeny i obrázky grafů v této práci.

3.3.3 libnetgraph

Dalším prostředkem pro komunikaci userspace procesů s moduly netgraphu je knihovna `libnetgraph(3)`. Je to knihovna napsaná v jazyce C a implementuje funkce nutné ke správě netgraphu. Samotná komunikace s netgraphem probíhá pomocí modulu `ng_socket(4)`. Prostředky této knihovny využívají právě nástroje `ngctl(8)` a `nghook(8)`.

3.4 Příklady využití Netgraphu

V adresáři `/usr/share/examples/netgraph` nalezneme příklady využití Netgraphu.

3.4.1 UDP tunel

Popíšeme si skript, který má za úkol vytvořit tunel mezi dvěma podsítěmi. Využívá se v něm tunelování IP protokolu přes UDP spojení. Přepis skriptu s českými komentáři:

```
#!/bin/sh

# Definice adres lokální a vzdálené vnitřní sítě,
# definice vnějších adres a číslo UDP portu,
# který bude použit pro tunel.
#
LOC_INTERIOR_IP=192.168.1.1
LOC_EXTERIOR_IP=1.1.1.1
REM_INTERIOR_IP=192.168.2.1
REM_EXTERIOR_IP=2.2.2.2
REM_INSIDE_NET=192.168.2.0
UDP_TUNNEL_PORT=4028

# Vytvoří uzel rozhraní ‘‘ng0’’, jestliže dosud neexistuje,
# jinak se pouze ujistí, zda není k něčemu připojen.
if ifconfig ng0 >/dev/null 2>&1; then
  ifconfig ng0 inet down delete >/dev/null 2>&1
  ngctl shutdown ng0:
else
  ngctl mkpeer iface dummy inet
fi

# Připojení UDP soketu k háku ‘‘inet’’ uzlu rozhraní za použití uzlu
# typu ng_ksocket(4).
#
ngctl mkpeer ng0: ksocket inet inet/dgram/udp
```

```
# Sváže (bind) UDP soket s lokální vnější IP adresou a portem.
#
ngctl msg ng0:inet bind inet/${LOC_EXTERIOR_IP}:${UDP_TUNNEL_PORT}

# Propojí UDP soket se vzdálenou vnější IP adresou a portem.
#
ngctl msg ng0:inet connect inet/${REM_EXTERIOR_IP}:${UDP_TUNNEL_PORT}

# Nakonfiguruje rozhraní typu point-to-point.
#
ifconfig ng0 ${LOC_INTERIOR_IP} ${REM_INTERIOR_IP}

# Vloží záznam do routovací tabulky, tak aby spojení do vzdálené vnitřní
# sítě probíhalo přes tunel.
#
route add ${REM_INSIDE_NET} ${REM_INTERIOR_IP}
```

3.4.2 Podvržení MAC adresy pomocí Netgraphu

Síťové adaptéry mají od výrobce pevně přidělenu tzv. hardwarovou MAC adresu. Tato adresa je unikátní (nebo by alespoň měla být). Tak jako se pomocí protokolu DNS spojují doménová jména s IP adresami, obdobně se na některých sítích vytváří pomocí protokolu ARP spojení mezi IP adresami a MAC adresami síťových rozhraní.

MAC adresa je na kartě většinou pevně „vypálena“ do paměti EPROM. Někdy je však třeba tuto adresu změnit, například pokud poskytovatel internetového připojení provádí mapování IP adres na MAC adresy a uživatel si chce např. jen na pár hodin připojit jiný počítač, aniž by musel absolvovat proceduru s odhlášením původní MAC adresy a registrací nové.

U některých síťových karet je možné nastavit MAC adresu softwarově pomocí nástroje `ifconfig(8)`. Ve skutečnosti nedojde ke změně pevně dané adresy, jen se karta tváří, jako by měla jinou. Co si však počít s kartami, které tuto softwarovou změnu neumožňují? V takovém případě nám může pomoci Netgraph.[13]

Následující příkazy vytvoří uzel typu `ng_bridge(4)` a virtuální ethernetové rozhraní.

- Odstraní IP adresu rozhraní.

  ```
  # ifconfig dc0 delete
  ```

- Vytvoří virtuální ethernetové rozhraní.

  ```
  # ngctl mkpeer . eiface hook ether
  ```

- Ověří, zda rozhraní existuje a MAC adresu má vynulovanou.

```
# ifconfig ngeth0
ngeth0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> mtu 1500
        ether 00:00:00:00:00:00
```

- Zapne virtuální ethernetové rozhraní.

```
# ifconfig ngeth0 up
ngeth0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet6 fe80::2d0:9ff:fe4c:9e5f%ngeth0 prefixlen 64 scopeid 0x4
        ether 00:00:00:00:00:00
```

- Pokud tak nebylo učiněno je potřeba zavést modul pro `ng_ether(4)`.

```
# kldload ng_ether
```

- Vytvoří most (bridge) a připojí dolní hák virtuálního rozhraní.

```
# ngctl mkpeer ngeth0: bridge lower link0
```

- Pojmenuje most.

```
# ngctl name ngeth0:lower mybridge
```

- Připojí hák dolní vrstvy fyzického rozhraní.

```
# ngctl connect dc0: mybridge: lower link1
```

- Připojí most na hák horní vrstvy fyzického rozhraní.

```
# ngctl connect dc0: mybridge: upper link2
```

- Připojí most na hák horní vrstvy virtuálního rozhraní.

```
# ngctl connect ngeth0: mybridge: upper link3
```

- Nastaví fyzické rozhraní, aby nepřepisovalo MAC adresu v hlavičkách rámců svou MAC adresou.

```
# ngctl msg dc0: setautosrc 0
```

- Přepne fyzické rozhraní do promiskuitního modu, tzn. karta předává do vyšších vrstev i pakety, které nejsou adresovány přímo pro ni.

```
# ngctl msg dc0: setpromisc 1
```

- Nastaví MAC adresu virtuálního rozhraní.

```
# ifconfig ngeth0 link 00:5c:16:10:dd:79
```

- Nechá si od DHCP serveru přidělit IP adresu pro virtuální rozhraní.

```
# dhclient ngeth0
# ifconfig ngeth0
ngeth0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet6 fe80::2de:adff:fe12:1212%ngeth0 prefixlen 64 scopeid 0x4
        inet 192.168.1.21 netmask 0xffffff00 broadcast 192.168.1.255
        ether 00:5c:16:10:dd:79
```

Schéma takto vytvořeného grafu je možno vidět na obrázku 3.1.

Kapitola 4

Další subsystémy využitelné pro modul

Pro návrh Netgraph modulu pro počítání statistik je potřeba porozumět několika dalším subsystémům a mechanismům jádra. V následujících kapitolách si představíme jednotlivé mechanismy. Nejprve danou problematiku uvedeme do širšího kontextu, potom se zaměříme na specifické detaily a nakonec danou oblast uvedeme do souvislosti s navrhovaným modulem.

4.1 Moduly jádra

Jádro systému FreeBSD se řadí mezi monolitická jádra, avšak jako většina moderních unixových systémů má podporu pro jaderné moduly. Pod pojmem jaderné moduly můžeme chápat dvě různé věci. Pokud chtěl vývojář do starších verzí BSD jádra (jako monolitického celku) přidat nějakou novou službu nebo subsystém, musel mít velice dobrou znalost interních jaderných mechanismů. To samozřejmě činilo vývoj obtížnějším a časově náročnějším. V novějších verzích nastal proces modularizace jádra, tedy rozbití na více logických celků, které poskytují jednotlivé služby. Tyto moduly jsou při zavádění systému seřazeny do logického sledu podle toho, jak na sobě závisí. Modularizované jádro umožňuje vývojářům snadněji přidávat nové služby a subsystémy.

Moduly můžeme rozdělit na dvě skupiny. Moduly, které mohou být načteny a naopak uvolněny z paměti v době běhu systému nazýváme *zaveditelné moduly jádra*. Moduly, které musejí být zavedeny do paměti při zavádění systému, nazýváme *permanentní jaderné moduly*. Tyto moduly poskytují základní služby jako správu paměti nebo plánovač procesů, a proto nemohou být odstraněny z paměti běžícího systému.

Programování zaveditelných modulů jádra je pro systémové programátory daleko pohodlnější a rychlejší. Programátor může modul zkompilovat, zavést, ladit pomocí debuggeru, uvolnit z paměti, změnit kus kódu, zkompilovat a znovu načíst bez toho, aniž by musel restartovat systém. Použití zaveditelných modulů jádra umožňuje také aktualizovat potřebné

části systému bez nutnosti restartu. Na druhou stranu při realizaci systému, který umožňuje zavádět a uvolňovat moduly jádra za běhu systému, vyvstává i otázka bezpečnosti. Ve starých verzích BSD bylo jádro kompletně imunní vůči změnám provedených uživatelem za doby běhu systému. Jediným rozhraním pro interakci s jádrem byla systémová volání. Uživatel sice mohl shodit svůj program, ale nemohl nijak ovlivnit jádro OS, pomineme-li možnost zneužití nějaké závažné chyby v jádře. Naproti tomu u jádra s podporou zaveditelných modulů hrozí nebezpečí, že uživatel, kterému se podaří získat práva superuživatele, může změnit některou část běžícího systému. Určité služby nemohou být uvolněny z běžícího jádra, což je sice určitá forma ochrany, ale modul, který je správně napsán může být kdykoliv zaveden, včetně těch podstrčených útočníkem. Určitým řešením bezpečnosti by byla podpora digitálních podpisů u jaderných modulů. Protože však FreeBSD digitálními podpisy modulů nedisponuje, dávají správci, kteří provozují FreeBSD v komerční sféře přednost spíše zakompilování všech potřebných služeb přímo do jádra a nepoužívají jaderné moduly, které mohou být načteny v době běhu systému. [3]

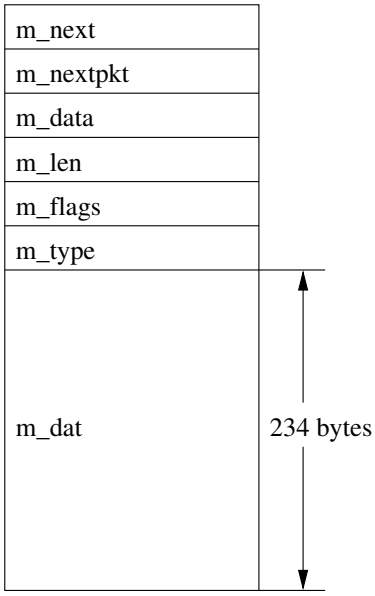
Celý subsystém Netgraphu můžeme mít k dispozici jako jaderný modul, nebo ho můžeme zakompilovat přímo do jádra, v tom případě je nutno přidat do konfiguračního souboru jádra řádek `options NETGRAPH`. Samotné moduly Netgraphu mají také formu jaderných modulů.

4.2 Mbuf

Požadavky na správu paměti pro prostředky meziprocesové komunikace (kam spadá i síťová komunikace) jsou trochu jiné než požadavky od ostatních částí jádra operačního systému. V obou případech jde sice o efektivní alokaci a uvolňování paměti, ale v případě meziprocesové komunikace je při implementaci různých protokolů často nutné paketům přidávat či odebírat různé hlavičky. Odesílaná data se často rozdělují do více paketů a naopak přijímané pakety se slučují do větších bufferů. Pakety jsou často řazeny do front, než jsou předány dál vyšším vrstvám pro příjem nebo nižším pro odeslání. Pro tyto účely existuje speciální správa paměti.

Alfou a omegou, kolem které je tato správa paměti vystavěna, je datová struktura *mbuf* (obr. 4.1). Paket je složen z řetězce těchto struktur (*mbuf chains*). Mbufy mají standardní velikost 256 bytů. Každý *mbuf* má na začátku hlavičku tvořenou strukturou *m_hdr*, ta obsahuje mimo jiné také ukazatel *m_next*, který ukazuje na další *mbuf* v řetězci. Hodnota NULL tohoto ukazatele značí, že je daný *mbuf* posledním v řetězci.

Za hlavičkou se nachází datová oblast, která je standardně velká 234 bytů (hlavička zabírá 22 bytů). Pokud by tato datová oblast přímo v *mbuf* svoji velikostí nestačila, alokuje se pro data externí datová struktura. V hlavičce *m_hdr* je proto ukazatel *m_data*, který ukazuje buď na datovou část *mbuf* nebo na externí datovou strukturu. Data se tedy nacházejí pouze přímo v *mbuf* nebo v externí struktuře, nikdy na obou místech. Dále je v hlavičce *m_hdr* k dispozici také údaj o velikosti datové oblasti, takže s jeho pomocí a s ukazatelem na data je datová oblast přesně vymezena.



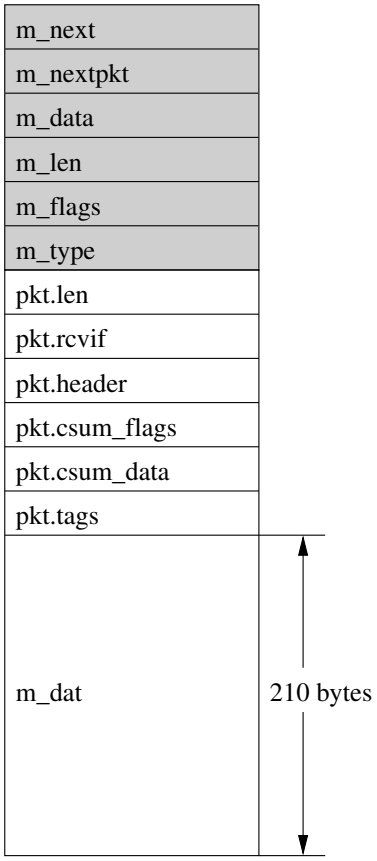
Obrázek 4.1: Datová struktura mbuf

Jak již bylo řečeno, může být několik struktur typu *mbuf* zřetězeno do útvaru nazývaného *mbuf chain*. Ty mohou být dále zřetězeny pomocí ukazatele *m_nextpkt* v hlavičce do dalšího seznamu objektů, který se v tomto případě nazývá fronta (*mbuf queue*).

Další zajímavou položkou v hlavičce *mbuf* je položka *flags*. Tyto „příznaky“ popisují jak samotný *mbuf*, tak i objekt uložený v *mbuf chain*. Příznaky popisují, zda má *mbuf* data uložena v externí datové struktuře (M_EXT), zda je k dispozici sekundární hlavička (M_PKTHDR). Pakety jsou uloženy v *mbuf chain* a první *mbuf* v řetězci má nastaven příznak M_PKTHDR. Dále můžou být nasteveny příznaky M_BCAST nebo M_MCAST, které značí, že daný paket byl vyslán jako broadcast nebo multicast, případně byl takto přijat.

Jestliže je nastaven příznak M_PKTHDR, následuje za běžnou hlavičkou *mbuf* hlavička paketu. Ta zmenší datovou oblast na 210 bytů (obr. 4.2). Hlavička paketu je přítomna pouze v počátečním *mbuf* v řetězci *mbuf chain* a obsahuje položku udávající celkovou velikost paketu, ukazatel na rozhraní, kterým byl daný paket přijat, ukazatel na hlavičku paketu, kontrolní součty a seznam doplňujících značek.

Mbuf nemusí mít data paketu obsažena přímo v sobě, ale může ukazovat na externí datovou strukturu. Data tak mohou být používána i v jiných částech síťového subsystému bez nutnosti vytvářet jejich kopie. Když je požadováno více kopií stejných dat, stačí když se *mbufy* odkazují na společná data. [3, 4]



Obrázek 4.2: Datová struktura mbuf s M_PKTHDR

4.2.1 Funkce pro manipulaci s mbufy

Pro alokaci nového *mbufu* se používá makro *MGET()*. Pokud chceme k nově vytvářenému *mbufu* připojit rovnou i hlavičku, můžeme použít makro *MGETHDR()*.

Funkce *m_adj()* upravuje vymezení datové oblasti v *mbufu*. Data nejsou nijak posouvána, pouze se manipuluje s odkazem *m_data* a hodnotou *m_len*. Při předávání paketu dalším vrstvám se tak můžeme posunout o hlavičku níž nebo výš.

Makro *mtod()* utvoří z ukazatele na *mbuf* hlavičku ukazatel na jemu příslušející datovou oblast přetypovaný na požadovaný typ. Analogicky makro *dtom()* vezme ukazatel na datovou oblast a vrací ukazatel na hlavičku *mbufu*, ale tato funkce funguje pouze, jsou-li data přímo součástí *mbufu*.

Funkce *m_pullup()* upraví *mbuf* tak, že požadovaný počet bytů je spojitě umístěn přímo v datové oblasti v *mbufu*. Tato operace se používá například pro zkoumaní protokolových hlaviček, abychom se na ně mohli podívat jako na spojitou datovou strukturu. [3]

4.3 Sysctl

Sysctl je systém pro sdílení proměnných mezi jádrem a uživatelskými procesy. Uživatel může tyto proměnné číst a některé i nastavovat pomocí utility `sysctl(8)` nebo pomocí jiného programu využívající knihovnu `sysctl(3)`.

Proměnné *sysctl* jsou řazeny do stromů a podstromů podobně jako adresářové struktury. Nejdůležitější podstromy první úrovně jsou tyto:

kern	jádro, procesy a limity
compat	vrstva kompatibility (s jinými operačními systémy)
vm	virtuální paměť
vfs	souborový systém
net	síť
debug	ladění
hw	hardware
machdep	závislé na architektuře
security	bezpečnost
user	user-level
p1003_1b	POSIX 1003.1B

Podstromy mohou obsahovat další podstromy nebo proměnné. Jednotlivé úrovně zanoření jsou v syntaxi utility `sysctl(8)` odděleny tečkou. Vezměme si jako příklad proměnnou *net.inet.ip.fw.enable*. Jejím nastavením můžeme zapnout, resp. vypnout firewall.

Následujícím příkazem si necháme zobrazit stručnou informaci k této proměnné.

```
# sysctl -d net.inet.ip.fw.enable
net.inet.ip.fw.enable: Enable ipfw
```

Další příkaz vypíše nastavení této proměnné.

```
# sysctl net.inet.ip.fw.enable
net.inet.ip.fw.enable: 1
```

Vídíme, že firewall je zapnutý. Tímto příkazem firewall vypneme:

```
# sysctl net.inet.ip.fw.enable=0
net.inet.ip.fw.enable: 1 -> 0
```

Statická deklarace

K statické deklaraci *sysctl* proměnných v kernelu slouží makra *SYSCTL_DECL()*. Takto vytvořené proměnné jsou inicializovány při inicializaci jaderného modulu a při jeho uvolnění jsou zničeny. K dispozici máme makra jako *SYSCTL_INT()* pro vytvoření položky typu celého čísla, *SYSCTL_LONG* pro dlouhé celé číslo, *SYSCTL_STRING()* pro řetězec znaků,

atd. Nový podstrom vytvoříme makrem *SYSCTL_NODE()*. Pokud některá proměnná při čtení nebo zápisu vyžaduje složitější zpracování dat k jejich zpřístupnění, můžeme vytvořit handler pro tuto proměnnou pomocí makra *SYSCTL_PROC()*. Proměnným je třeba při vytváření nastavit práva (pouze pro čtení, pro čtení i zápis rootem, pro čtení i zápis libovolným uživatelem, atd.). Podrobnější popis těchto maker a několik ukázkových příkladů najdeme v manuálové stránce k `sysctl(9)`.

Dynamická deklarace

Statická alokace *sysctl* proměnné nám umožní, že se proměnná vytvoří, když modul načteme do jádra. Pokud však musíme vytvářet proměnné během běhu programu modulu, musíme sáhnout po funkci *sysctl_add_oid()*. Tato funkce vytvoří obecně jakýkoliv dostupný typ proměnné *sysctl*. Spíše je však výhodnější použít připravená makra pro jednotlivé typy (*SYSCTL_ADD_NODE(), SYSCTL_ADD_INT(), SYSCTL_ADD_UINT()*, apod.), kód potom bude alespoň o něco čitelnější.

U staticky deklarovaných *sysctl* proměnných nemáme žádnou možnost, jak je zrušit, kromě odstranění modulu z paměti. Dynamicky deklarované proměnné i celé podstromy můžeme rušit pomocí funkce *sysctl_remove_oid()*. K přesouvání objektů na jiné podstromy slouží funkce *sysctl_move_oid()*.

Podrobnější popis těchto funkcí a příklady použití nalezneme v manuálové stránce `sysctl_add_oid(9)`.

Při vytváření nových proměnných je dobré si uvědomit, že tyto proměnné mohou být používány přímo uživateli, knihovnami, programy nebo mohou být uváděny v dokumentaci. Názvy podstromů a proměnných *sysctl* bychom měli vybírat tak, abychom vyloučili případné kolize se současnými názvy a měli bychom také myslet na budoucnost a vybírat zvláště pro nové podstromy vhodná jména.

Kapitola 5

Modul pro počítání statistik

Jedním z úkolů této bakalářské práce bylo vytvořit netgraphový modul pro počítání síťových statistik. Jako postačující se ukázala koncepce modulu se dvěma háky, který bude po zapojení počítat přes něj tekoucí data (počet přenesených paketů a jejich celkovou velikost).

Jak již bylo řečeno, Netgraph má do jisté míry objektově orientovaný charakter. Při implementaci vlastního uzlu se používá přetížení generických metod vlastními funkcemi. Nejdůležitějšími metodami v uzlu je funkce pro příjem dat a funkce zpracování kontrolních zpráv.

Funkce pro příjem dat má přístup k datům jako ke strukturám typu *mbuf*. Při přenosu datových zpráv mezi uzly nedochází ke zbytečnému kopírování, ale namísto toho se přenášejí pouze ukazatele na *mbufy*. Tyto ukazatele na *mbufy* jsou ve skutečnosti obaleny strukturou nazývanou *item*. Ta obsahuje položky jako adresu cílového uzlu, různé příznaky apod. Tyto implementační detaily samotného Netgraphu nás však nemusí příliš zajímat. Podstatné je, že se dostaneme na strukturu *mbuf*, kterou můžeme dále zkoumat.

Pro získání naměřených dat z modulu je možné využít mechanismu kontrolních zpráv. Funkce pro příjem kontrolních zpráv při příjmu daného typu zprávy odešle jako odpověď strukturu s naměřenými daty. Další možností jak získávat naměřená data z modulu je využití systému *sysctl*.

5.1 Inspirace v ng_tee

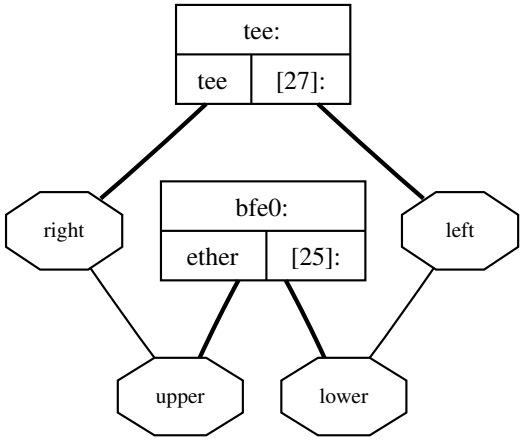
Pro základní počítání průchozích dat se dá využít modul `ng_tee(4)`. Uzel tohoto typu si pro každý hák udržuje jednoduché statistiky o přijatých a odeslaných datech. Pro jednoduché účtování na linkové vrstvě můžeme připojit uzel typu *tee* mezi háky *lower* a *upper* uzlu *ether*.

- Modul `ng_ether(4)` je potřeba načíst manuálně, většina ostatních modulů se zavádí automaticky při vytváření prvního uzlu daného typu.

  ```
  # kldload ng_ether
  ```

- Modul `ng_ether(4)` vytvořil pro každé ethernetové rozhraní odpovídající uzel. Dejme tomu, že chceme účtovat data na zařízení `bfe0`. Připojíme na uzel s názvem "bfe0" nový uzel *tee* a pojmenujeme ho ASCII názvem "tee".

```
# ngctl mkpeer bfe0: tee lower left
# ngctl connect bfe0: lower upper right
# ngctl name bfe0:lower tee
```



Obrázek 5.1: Připojení uzlu tee na ether

- Vytvořený graf můžeme vidět na obrázku 5.1. Uzlu "tee" nyní pošleme zprávu, aby vrátil naměřené statistiky.

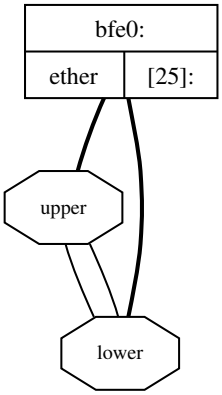
```
# ngctl msg tee: getstats
Rec'd response ``getstats'' (1) from ``[27]:'':
Args:   { right={ inOctets=32298 inFrames=376 outOctets=2303817
outFrames=27181 } left={ inOctets=2303817 inFrames=27181
outOctets=32298 outFrames=376 } }
```

- Jestliže chceme uzel "tee" odstranit, pošleme mu generickou kontrolní zprávu *shutdown*.

```
# ngctl shutdown tee:
```

- Povšimněme si však, že došlo ke spojení háků *lower* a *upper* (viz. obr. 5.2). To je totiž specialita `ng_tee(4)`, že pokud jsou připojeny háky *left* a *right*, dojde při vypnutí uzlu ke spojení jeho protějšků. Uzel tee se tak odpojí z cesty a mezeru po sobě zacelí.

- To, že zůstanou háky *lower* a *upper* typu ether spojené, nemá na jeho funkci žádný vliv. Pokud bychom tomu chtěli předejít, stačí jeden z háků uzlu tee nejdříve odstranit a



Obrázek 5.2: Po vypnutí uzlu tee

potom uzel vypnout. Další možností je odstranit oba háky. Potom dojde jako u většiny modulů po odstranění všech háků k automatickému vypnutí uzlu.

```
# ngctl rmhook tee: left
# ngctl rmhook tee: right
```

- Uzly typu *ether* nelze jako ostatní uzly vypnout pomocí kontrolní zprávy *shutdown*. Namísto toho lze tyto uzly odpojit od ethernetových zařízení zprávou *detach*.

```
# ngctl msg bfe0: detach
```

- Moduly je potom možné odstranit z paměti:

```
# kldunload ng_tee
# kldunload ng_ether
```

5.2 První verze modulu

Modul pro počítání statistik byl pojmenován jako `ng_stat`. První verze modulu vznikla zjednodušením modulu `ng_tee(4)`. Modul `ng_tee(4)`, stejně jako celý Netgraph je zveřejněn pod BSD licencí, takže nic nebránilo tomu vycházet při implementaci z kódu tohoto modulu.

Uzel typu *tee* má háky *right2left* a *left2right*, na které posílá duplikovaná data. To se ukázalo jako zbytečné a proto byly tyto háky odstraněny. Zbytečné je také, aby se pro každý hák počítala příchozí i odchozí data. V případě že jsou data posílána na protější hák, stačí aby se zaznamenávala např. pouze data přicházející do uzlu.

Po připojení na uzel typu *ether* počítala první verze modulu jednotlivé ethernetové rámce a sčítala jejich velikosti. Velikost přijatého rámce se získá z položky *pkt.len* v hlavičce

prvního *mbufu* v řetězci *mbuf chain*. To, že naměřená velikost odpovídá velikosti ethernetových rámců, bylo ověřeno srovnáním výstupu s výstupem softwarového síťového analyzátoru Wireshark (`http://www.wireshark.org`).

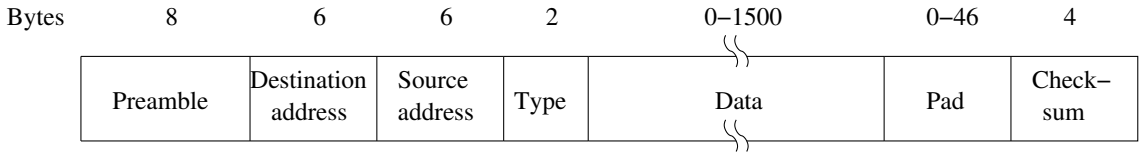
Koncepce sledování ethernetových rámců je použitelná i na bezdrátové LAN (Wi-Fi), protože tam se k přenosu používá také ethernetový protokol ([8, str. 69]).

5.3 Rozšíření funkčnosti

Rozšíření modulu oproti první verzi spočívá ve zkoumání struktury přenášených ethernetových rámců. Při zkoumání *mbufu* postupujeme vždy podle tohoto schématu:

- Nejprve, pokud je třeba, upravíme *mbuf* pomocí funkce *m_pullup()*, abychom měli jistotu, že se požadovaný počet bytů nachází spojitě v datové oblasti *mbufu*.
- Poté získáme ukazatel na data pomocí makra *mtod()*.
- A potom se už můžeme na data podívat jako na požadovanou strukturu.

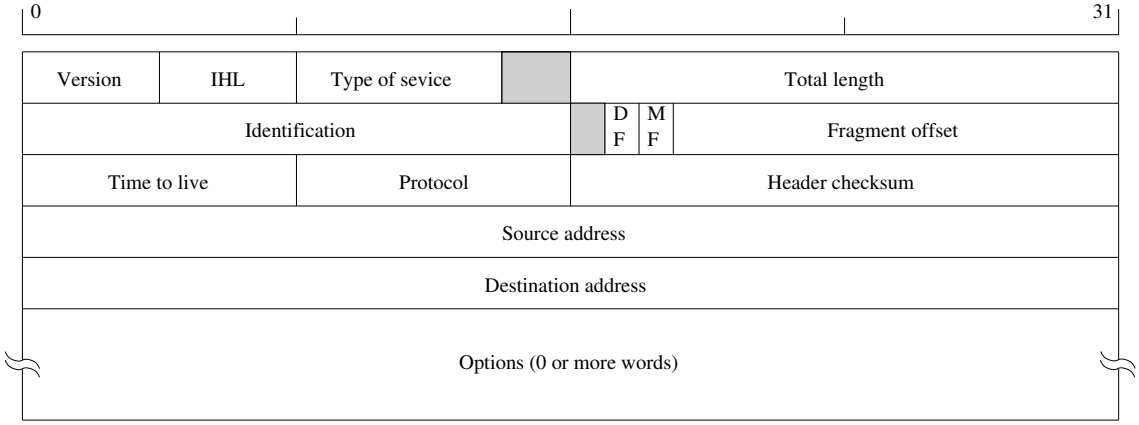
Jako první se ke zkoumání na linkové vrstvě nabízejí ethernetové rámce. Znázornění jednotlivých položek v ethernetovém rámci vidíme na obr. 5.3. Zajímavá je položka type, která udává typ daného spojení. Po porovnání této položky s konstantami z hlavičkového souboru `net/ethernet.h` můžeme rozhodnout, do jakého protokolu na síťové vrstvě spojení patří. Implementovaný modul sleduje protokoly IP a ARP. O další protokoly je možné modul rozšířit jednoduchým zásahem do zdrojového kódu.



Obrázek 5.3: Ethernetový rámec IEEE 802.3

Inkrementací ukazatele na datovou oblast se můžeme posunout v ethernetovém rámci za položku type. To nám umožní podívat se na data z perspektivy protokolu na vyšší vrstvě. Na obrázku 5.4 vidíme strukturu hlavičky IP protokolu verze 4. Položka *Protocol* obsahuje informaci o protokolu transportní vrstvy daného paketu. Definice konstant jednotlivých protokolů najdeme v hlavičkovém souboru `netinet/in.h`. Nad protokolem IP rozpoznává náš modul protokoly TCP, UDP a ICMP.

Podobným způsobem si lze všímat i dalších položek v IP hlavičce. Musíme si však dát pozor na to, že jednotlivé byty položek v IP hlavičce jsou uloženy v pořadí Big-Endian, což je standard v síťové komunikaci. K převodu čísel mezi kódováním naší architektury a pořadím Big-Endian na síti slouží makra *htonl()* pro 32-bitová celá čísla a *htons()* pro 16-bitová celá čísla. Pro opačný převod ze síťového pořadí bytů na pořadí naší architektury existují



Obrázek 5.4: Hlavička protokolu IPv4

makra *ntohl()* a *ntohs()*. Na architekturách s pořadím bytů Big-Endian jsou tato makra definována tak, že s předanými hodnotami nic neprovádějí, přesto (nebo snad právě proto) se doporučuje je používat, neboť se tak zvýší přenosnost našeho programu. Ve zdrojovém kódu implementovaného modulu je pro ukázku u protokolu TCP nastaveno účtování pouze spojení s určitou pevně nastavenou IP adresou.

Dále je možné posunout se o velikost IP hlavičky a zkoumat hlavičku dalšího protokolu, např. TCP nebo UDP, a zaznamenávat jen spojení určitého portu apod. V implementovaném modulu je toho využito a pro demonstraci se účtují v TCP sekci pouze spojení asociovaná s určitým napevno nastaveným portem.

5.4 Komunikace s uzlem

Pro každý z protokolů IP, ARP, TCP, UDP a ICMP jsou v modulu zvlášť počítadla pro tok dat ze sítě (*upstream* – z nižších vrstev do vyšších) a pro opačný směr (*downstream*). Tyto naměřené hodnoty je možné získat dvojím způsobem:

1. Pomocí kontrolních zpráv.
2. Přes systém sysctl.

Pro oba způsoby se používají jedny a ty samé proměnné. To, že mohou být použity oba systémy, nijak neovlivňuje výkon modulu. Konkrétní popis kontrolních zpráv a sysctl proměnných je uveden v příloze C v návodu k použití.

Kapitola 6

Závěr

6.1 Srovnání s podobnými moduly

Na závěr se nabízí srovnání s ostatními podobnými nástroji na sledování síťového provozu a vytváření statistik. Podobným systémem je například Cisco NetFlow.

6.1.1 Cisco NetFlow

Cisco NetFlow je protokol pro sledování a sbírání informací o toku na síti. Implementace tohoto protokolu sestává ze dvou od sebe oddělených programů. Tou první částí jsou senzory, které rozpoznávají jednotlivé „toky“ (flows) v síťovém provozu. Toky jsou rozlišovány podle IP adres koncových bodů, TCP/UDP portů, ToS a vstupních rozhraní. Naměřená data jsou posílána přes UDP spojení dalším programům, tzv. sběračům. Sběrače skladují naměřená data, ze kterých pak můžou být vytvářeny různé grafy apod. Senzory a sběrače mohou být nainstalovány odděleně na různých strojích na síti. Ve FreeBSD existuje senzor pro NetFlow implementovaný v Netgraphu. Je jím modul `ng_netflow(4)`. [7]

NetFlow sleduje datová spojení na síťové vrstvě protokolu IP, modul `ng_stat` naproti tomu může sledovat spojení už na nižší linkové vrstvě a vidí tak protokoly, které jsou i mimo IP.

6.2 Návrhy na rozšíření

Modul `ng_stat` by se dal samozřejmě dále rozšířit. Jednou z oblastí rozšíření by mohla být hlubší analýza procházejících paketů a rozpoznávání více protokolů. Rozšíření by nebylo nijak zvláště složité, stačí postupovat způsobem naznačeným v kapitole 5.

Další rozšíření práce by mohlo spočívat ve vytvoření systémového utility operující v user-space. Nástroj by mohl využívat knihovnu `netgraph(3)` v případě, že by komunikoval s modulem přes kontrolní zprávy Netgraphu. V případě komunikace přes *sysctl*, by bylo vhodné použít knihovnu `sysctl(3)`.

Namísto stávajícího přizpůsobování si modulu přímo úpravou zdrojového kódu, by bylo vhodné vytvořit systém konfigurace, resp. nějaký konfigurační jazyk.

Dobré by také bylo provést testování vlivu modulu na výkonnost systému a provedení případných optimalizací.

Seznam použitých zdrojů

[1] Cobbs, A.: All About Netgraph. [online], 2000, [cit. 2007-04-24].
URL http://ezine.daemonnews.org/200003/netgraph.html

[2] Elischer, J.; Cobbs, A.: *FreeBSD Kernel Interfaces Manual: NETGRAPH(4)*. FreeBSD, 2004, [rev. 2004-06-01].

[3] McKusick, M. K.; Neville-Neil, G. V.: *The Design and Implementation of the FreeBSD Operating System*. Addison-Wesley Professional, 2005, ISBN 0-201-70245-2.

[4] Patočka, M.: Porovnání systémů Linux a FreeBSD (13). [online], 2004, [cit. 2007-05-01].
URL http://www.root.cz/clanky/porovnani-linux-freebsd-13/

[5] Raymond, E. S.: *The Art of Unix Programming*. Addison-Wesley Professional, 2003, ISBN 0-13-142901-9.
URL http://www.faqs.org/docs/artu/

[6] Ritchie, D. M.: The Evolution of the UNIX Time-sharing System. *BSTJ*, ročník 63, 8, 1984: s. 1577–1594.
URL http://cm.bell-labs.com/cm/cs/who/dmr/hist.html

[7] Smirnoff, G.: *FreeBSD Kernel Interfaces Manual: NG_NETFLOW(4)*. FreeBSD, 2006, [rev. 2006-03-02].

[8] Tanenbaum, A. S.: *Computer Networks*. Prentice Hall, 2003, ISBN 0-13-066102-3.

[9] The FreeBSD Documentation Project: FreeBSD Handbook. [online], 2007, [cit. 2007-05-05].
URL
http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html

[10] The Jargon File, version 4.4.7: Copycenter. [online], [cit. 2007-05-05].
URL http://catb.org/~esr/jargon/html/C/copycenter.html

[11] Wikipedia: Berkeley Software Distribution. [online], 2007, [rev. 2007-04-29], [cit. 2007-05-04].
URL http://en.wikipedia.org/wiki/Berkeley_Software_Distribution

[12] Wikipedie: BSD licence. [online], 2007, [rev. 2007-04-15], [cit. 2007-05-08].
URL http://cs.wikipedia.org/wiki/BSD_licence

[13] Yeske, D.: MAC address spoofing on FreeBSD using netgraph. [online], 2004, [cit. 2007-04-24].
URL http://ezine.daemonnews.org/200406/netgraph.html

Seznam příloh

Příloha A

BSD licence

Neoficiální český překlad dnešní podoby licence[12]:

> Copyright © ROK, VLASTNÍK PRÁV. Všechna práva vyhrazena.
>
> Redistribuce a použití zdrojových i binárních forem díla, v původním i upravovaném tvaru, jsou povoleny za následujících podmínek:
>
> - Šířený zdrojový kód musí obsahovat výše uvedenou informaci o copyrightu, tento seznam podmínek a níže uvedené zřeknutí se odpovědnosti.
> - Šířený binární tvar musí nést výše uvedenou informaci o copyrightu, tento seznam podmínek a níže uvedené zřeknutí se odpovědnosti ve své dokumentaci a/nebo dalších poskytovaných materiálech.
> - Ani jméno vlastníka práv, ani jména přispěvatelů nemohou být použita při podpoře nebo právních aktech souvisejících s produkty odvozenými z tohoto software bez výslovného písemného povolení.
>
> TENTO SOFTWARE JE POSKYTOVÁN DRŽITELEM LICENCE A JEHO PŘISPĚVATELI JAK STOJÍ A LEŽÍ A JAKÉKOLIV VÝSLOVNÉ NEBO PŘEDPOKLÁDANÉ ZÁRUKY VČETNĚ, ALE NEJEN, PŘEDPOKLÁDANÝCH OBCHODNÍCH ZÁRUK A ZÁRUKY VHODNOSTI PRO JAKÝKOLIV ÚČEL JSOU POPŘENY. DRŽITEL, ANI PŘISPĚVATELÉ NEBUDOU V ŽÁDNÉM PŘÍPADĚ ODPOVĚDNI ZA JAKÉKOLIV PŘÍMÉ, NEPŘÍMÉ, NÁHODNÉ, ZVLÁŠTNÍ, PŘÍKLADNÉ NEBO VYPLÝVAJÍCÍ ŠKODY (VČETNĚ, ALE NEJEN, ŠKOD VZNIKLÝCH NARUŠENÍM DODÁVEK ZBOŽÍ NEBO SLUŽEB; ZTRÁTOU POUŽITELNOSTI, DAT NEBO ZISKŮ; NEBO PŘERUŠENÍM OBCHODNÍ ČINNOSTI) JAKKOLIV ZPŮSOBENÉ NA ZÁKLADĚ JAKÉKOLIV TEORIE O ZODPOVĚDNOSTI, AŤ UŽ PLYNOUCÍ Z JINÉHO SMLUVNÍHO VZTAHU, URČITÉ ZODPOVĚDNOSTI NEBO PŘEČINU (VČETNĚ NEDBALOSTI) NA JAKÉMKOLIV ZPŮSOBU POUŽITÍ

TOHOTO SOFTWARE, I V PŘÍPADĚ, ŽE DRŽITEL PRÁV BYL UPOZORNĚN NA MOŽNOST TAKOVÝCH ŠKOD.

V původní licenci licenci od University of California in Berkeley (tzv. staré BSD licenci) byl obsažen navíc tento bod:

- Všechny propagační materiály zmiňující vlastosti nebo použití tohoto softwaru musejí obsahovat následující text:

  Tento produkt zahrnuje software vytvořený VLASTNÍKEM PRÁV a přispěvatelů.

Příloha B

Návod pro překlad a instalaci modulu

Pro sestavení modulu je potřeba mít k dispozici zdrojové kódy jádra. Nejpohodlnější je k jejich instalaci použít nástroj `sysinstall(8)`. Jako uživatel *root* spustíme `sysinstall`, z nabídky postupně vybereme *Configure*, *Distributions*, *src* a zaškrtneme položky *base* a *sys*. Popisy alternativních způsobů instalace zdrojových kódů jádra najdeme například v [9, kap. 8.3].

Překlad spustíme příkazem `make`. Přeložený modul je potom potřeba nakopírovat do adresáře `/boot/kernel/` nebo alespoň do něj umístit symbolický link na modul.

```
$ make
```

a

```
# cp ng_stat.ko /boot/kernel/
```

nebo

```
# ln -s ng_stat.ko /boot/kernel/
```

Příloha C

Návod k použití modulu

Pro snadné načtení modulu a připojení uzlu na požadovaná ethernetová rozhraní byl vytvořen shellový skript `etherstat.sh`. Jeho použití je následující:

- Pokud je třeba, upravíme proměnnou `IFACES`, která definuje rozhraní, na kterých si přejeme sledovat provoz.
- Skript spustíme s parametrem `start`:

  ```
  # etherstat.sh start
  Loading ng_stat.ko...done
  ```

- Pro odpojení uzlů z grafu a uvolnění modulu z paměti slouží parametr `stop`:

  ```
  # etherstat.sh stop
  Unloading ng_stat.ko...done
  ```

- Navíc je k dispozici parametr `restart`, který spustí postupně sekce `stop` a `start`.

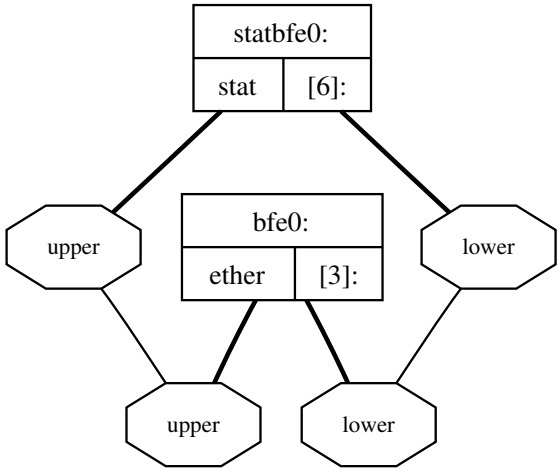
Na obrázku C.1 vidíme připojení modulu na uzel ethernetového rozhraní.

C.0.1 Kontrolní zprávy

Modul rozumí následujícím kontrolním zprávám:

Binární formát	Textový formát	Význam
`NGM_STAT_GET_STATS`	`getstats`	získá statistiky
`NGM_STAT_CLR_STATS`	`clrstats`	smaže statistiky (vynuluje počítadla)
`NGM_STAT_GETCLR_STATS`	`getclrstats`	atomicky získá a smaže statistiky

Vypsání naměřených statistiky docílíme s nástrojem `ngctl(8)` např. takto:



Obrázek C.1: Připojení uzlu stat na ether

```
# ngctl msg statbfe0: getstats
Rec'd response ``getstats'' (1) from ``[18]:'':
Args:   { downstream={ ip_octets=19150259 ip_frames=348114
udp_octets=60071 udp_frames=402 arp_octets=294 arp_frames=7 }
upstream={ ip_octets=781547454 ip_frames=541149 udp_octets=1518023
udp_frames=10125 arp_octets=2618880 arp_frames=43648 } }
```

C.0.2 SYSCTL

Příklad výpisu všech statistik všech uzlů typu *stat* pomocí `sysctl(8)`:

```
net.stat.00000006.name: statbfe0
net.stat.00000006.downstream.ip_octets: 19347667
net.stat.00000006.downstream.ip_frames: 350985
net.stat.00000006.downstream.tcp_octets: 0
net.stat.00000006.downstream.tcp_frames: 0
net.stat.00000006.downstream.udp_octets: 65085
net.stat.00000006.downstream.udp_frames: 430
net.stat.00000006.downstream.icmp_octets: 0
net.stat.00000006.downstream.icmp_frames: 0
net.stat.00000006.downstream.arp_octets: 336
net.stat.00000006.downstream.arp_frames: 8
net.stat.00000006.upstream.ip_octets: 786816119
net.stat.00000006.upstream.ip_frames: 546585
net.stat.00000006.upstream.tcp_octets: 0
net.stat.00000006.upstream.tcp_frames: 0
net.stat.00000006.upstream.udp_octets: 1719152
```

```
net.stat.00000006.upstream.udp_frames: 11582
net.stat.00000006.upstream.icmp_octets: 0
net.stat.00000006.upstream.icmp_frames: 0
net.stat.00000006.upstream.arp_octets: 3078900
net.stat.00000006.upstream.arp_frames: 51315
```

Jak můžeme vidět z předchozího výpisu, uzel si v `net.stat` vytvoří podstrom, jehož jménem je adresa uzlu v hexadecimálním tvaru. Je to z toho důvodu, že vytvořený uzel nemusí vůbec dostat jméno. Pokud ale uzel jméno dostane, objeví se v proměnné `name`.