VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

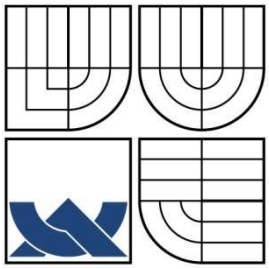
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

MĚŘENÍ INTENZITY PROVOZU BĚHEM PEVNĚ DANÝCH INTERVALŮ V AP

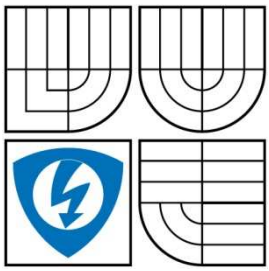
DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE Bc. PAVEL KUBÍK
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

MĚŘENÍ INTENZITY PROVOZU BĚHEM PEVNĚ DANÝCH INTERVALŮ V AP

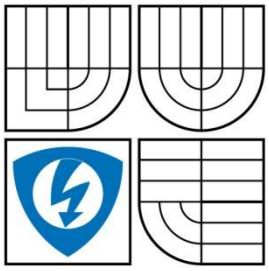
MEASUREMENTS OF THE INTENSITY OF TRAFFIC WITHIN A FIXED INTERVAL OF THE AP

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE — BC. PAVEL KUBÍK
AUTHOR

VEDOUCÍ PRÁCE — ING. TOMÁŠ MATOCHA
SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Pavel Kubík	**ID:**	98639
Ročník:	2	**Akademický rok:**	2010/2011

NÁZEV TÉMATU:

Měření intenzity provozu během pevně daných intervalů v AP

POKYNY PRO VYPRACOVÁNÍ:

Zrealizujte měření provozu a to pomocí vhodného způsobu. To sestává z těchto kroků: Využijte vhodný bezdrátový přístupový bod standardu IEEE 802.11, který umožní nízkoúrovňový přístup k softwarovým komponentám a vlastnostem přístupového bodu. Mělo by být možno rozlišovat mezi jednotlivými spojeními a následně získat data ke statistickému zpracování. Nejvhodnější je použití C/C++ programovacího jazyka. Důležitá je především vhodná interpretace dat, nejlépe pomocí dynamických grafů. Zároveň zvažte možnost využití některých grafických knihoven jako je qt nebo tcl/tk.

DOPORUČENÁ LITERATURA:

[1] NEMETH, E., SNYDER, G., HEIN T. Linux - Kompletní příručka administrátora. Computer Press, 2004. 880 s. ISBN: 80-722-6919-4.
[2] PRATA, S. Mistrovství v C++. Computer Press, 2007. 1120 s. ISBN: 978-80-251-1749-1

Termín zadání:	07.02.2011	**Termín odevzdání:**	26.5.2011

Vedoucí práce: Ing. Tomáš Matocha

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

Abstrakt

Práce se zabývá analýzou síťového provozu na směrovači s otevřeným firmware. Nejprve jsou vybrány softwarové platformy, na základě kompatibility s dostupnými přístroji. Poté jsou zhodnoceny vlastnosti nutné pro vývoj vlastní aplikace. Podpora různých programovacích jazyků, podporované vývojové prostředí SDK a dostupné moduly nebo knihovny pro práci se síťovým rozhraním. Na základě těchto faktorů je následně zvolen způsob realizace požadovaného programu. Ten je realizován na firmwaru OpenWRT v jazyce C/C++ s využitím síťové knihovny pcap. Těmito prostředky je zachytáván a analyzován síťový provoz. Získaná data jsou zpracována pomocí metod z technické analýzy jmenovitě na základě klouzavých průměrů, Stochastického oscilátoru a Bollingerových pásem. Z těchto výsledků jsou vytvořeny a ověřovány odhady vývoje provozu. Ty jsou postaveny na lineární extrapolaci, zjednodušené pro pevně dané intervaly. Platnost každé metody je ověřována na základě toho, zda odhaduje hodnoty objemu provozu v rámci Bollingerova pásma, daného standardní odchylkou. Každá metoda je několikanásobně testována v reálném provozu s různými vstupními parametry. Poté je vyhodnocen vliv parametrů na chybovost metod. Jednotlivé metody jsou srovnány a zhodnoceny na základě chování v různých scénářích a podle průměrné relativní chyby.

Klíčová slova

OpenWRT, BSD Packet Filter, pcap, Exponenciální klouzavý průměr, Stochastický oscilátor, Bollingerovo pásmo, Extrapolace

Bibliografická citace této práce

KUBÍK, P. *Měření intenzity provozu během pevně daných intervalů v AP.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 49 s. Vedoucí diplomové práce Ing. Tomáš Matocha.

Abstract

The thesis analyzes the network traffic on a router with open source firmware. First is chosen a software platform, based on compatibility with available equipment. Then are assessed properties necessary for the development of custom applications. Support for various programming languages provided by the SDK, development environment and the available modules and libraries, for working with network interface. Based on these factors is then chose method to realize the program. He is implemented on the OpenWRT firmware in C / C + + using network library pcap. These funds are used to capture and analyze network traffic. Obtained data are processed using methods of technical analysis, namely on the basis of moving averages, stochastic oscillator and Bollinger bands. Based on results of these methods are generated and verified estimates of traffic. They are based on linear extrapolation, simplified for fixed intervals. The validity of each method is verified on base of the estimated value. Method is verified if estimated value of the traffic volume is in the Bollinger band, which is given by the standard deviation. Each method is tested several times in real traffic with different input parameters. Then is evaluated the influence of parameters on the error rate of methods. Individual methods are compared and evaluated based on the behavior in different scenarios and based on the average relative error.

Keywords

OpenWRT, BSD Packet Filter, pcap, Exponential Moving Average, Stochastic Oscillator, Bollinger bands, Extrapolation

Bibliographic citation of this thesis

KUBÍK, P. *MEASUREMENTS OF THE INTENSITY OF TRAFFIC WITHIN A FIXED INTERVAL OF THE AP.* Brno: Brno University of Technology, The Faculty of Electrical Engineering and Communication, 2011. 49 p. Thesis supervisor Ing. Tomáš Matocha.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma „**Měření intenzity provozu během pevně daných intervalů v AP“** jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ……. .....……

podpis autora

OBSAH

1 Úvod

Vývoj hardware v současné době umožňuje stále více možností využití směrovačů. V této práci bude sledováno a interpretováno provozní zatížení na koncovém přístupovém bodu, který je určen pro potřebu běžných uživatelů. V současné době existuje velké množství aplikací pro monitorování, které obvykle využívají sondy a s nimi související protokol NetFlow. Takovéto řešení je primárně určeno pro informování administrátora o stavu vzdáleného prvku. Účelem této práce je vybrat vhodné hardwarové a softwarové prostředky, které umožní vytváření statistiky a následný odhad vývoje provozu přímo na směrovači. Tyto informace mohou být dále použity především pro automatizované filtrování uživatelů, usměrňování provozu nebo pro iniciaci handoveru.

Růst výkonu síťových prvků umožnil poskytování stále více služeb, které byly dřív vyhrazeny pouze serverům. Také pokrytí bezdrátovými sítěmi prudce stouplo, což se nakonec projevilo i v návrhu přenosových protokolů. Ty v poslední době mohou podporovat handovery mezi bezdrátovými počítačovými a telekomunikačními sítěmi, jako protokol IEEE 802.21. Na síťových prvcích se tak projevila potřeba získávání parametrů, které byli dříve typické pro telekomunikační prvky, například celkové zatížení jednotlivých rozhraní.

Cílem práce je právě sledování síťového rozhraní, na přístupovém směrovači v bezdrátové síti. Provoz bude vyhodnocován vlastním realizovaným programem na zvolené softwarové platformě a hardwaru. Metody zpracování získaných dat by měli umožnit určovat vývoj provozu v následujícím měřeném intervalu.

V první části bude proveden výběr hardware a souvisejících softwarových prostředků. Přesto, že lepší směrovače dnes běží na jádru linuxu, je jen málo výrobců kteří používají otevřený systém. Je proto nutné zvolit takové řešení, které umožní co nejméně omezený vývoj vlastních aplikací. Zároveň musí poskytovat vhodné prostředky pro vytvoření aplikace s požadovanými vlastnostmi.

Na základě zvoleného řešení bude následně vytvořen program zachytávající síťový provoz na bezdrátovém rozhraní. Ten bude sledovat přenesené objemy dat ve směru od uživatelů i k uživatelům. Tyto údaje budou statisticky zpracovány a vhodně interpretovány, pro charakterizaci provozu. Dále budou realizovány některé mechanizmy ze statistiky a technické analýzy. Ty budou společně sloužit jako základ, pro vytvoření několika způsobů odhadu síťového provozu. Vytvořené metody odhadu budou na závěr otestovány v reálném provozu a bude zvolena nejúspěšnější metoda.

2 TEORETICKÝ ROZBOR

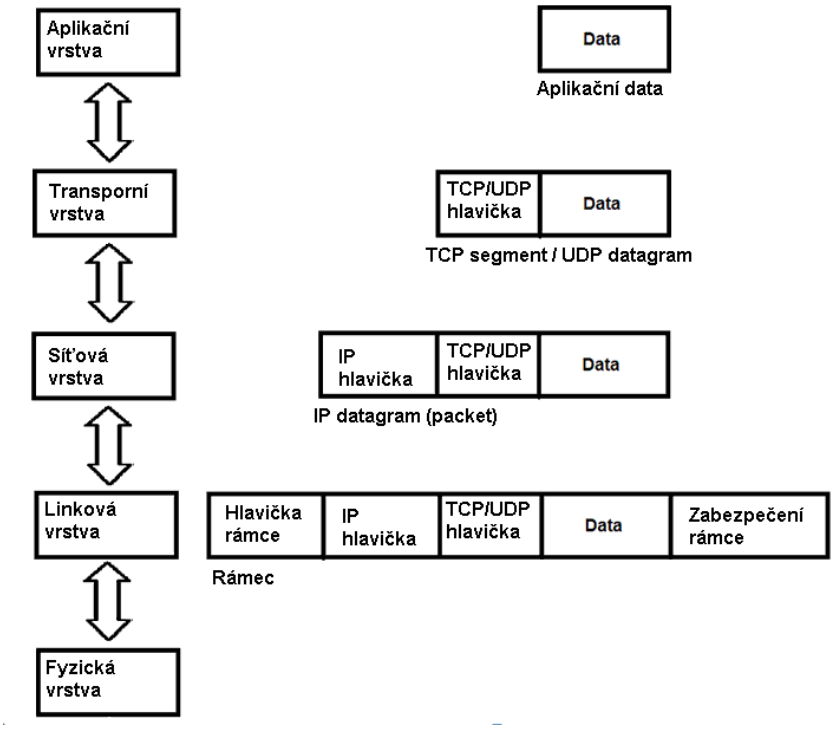
V této části bude rozebrána problematika, která je nutná pro pochopení činností hardwaru i softwaru při zpracování dat ze síťových rozhraní. Budou popsány základní mechanizmy zachytávání, způsoby zpracování a související protokoly a systémové prostředky.

2.1 IEEE 802.11

IEEE 802.11 je původním standardem pro bezdrátovou komunikaci, který se dnes již nevyužívá, ale poprvé definoval podobu fyzické vrstvy. Všechny další varianty standardu 802.11 mají stejnou vrstvovou strukturu, i když mohou využívat rozdílně rozprostřené spektrum, podporovat vyšší rychlosti či diverzitu. O přístup k fyzické vrstvě se starají ovladače a ne vždy je poskytnuta možnost číst stavové informace. Vyšším vrstvám se bezdrátový adaptér jeví stejně jako běžné síťové rozhraní. Proto jsou většinou data od rozhraní zapouzdřena jako Ethernetové rámce. Z hlediska zachytávání provozu tak téměř není rozdíl mezi daty od drátového a bezdrátového síťového rozhraní.

2.2 Zapouzdření

Než je možné poslat aplikační data na síťové rozhraní, musí být přizpůsobeny pro všechny protokoly, které budou použity při jejich přenášení. Tomuto procesu se říká zapouzdření (encapsulation). Každý protokol, z dané sady, si k datům přidá vlastní záhlaví a někdy i zápatí.



Obr. 2.1: Zapouzdření na jednotlivých vrstvách modelu TCP/IP

Při potřebě odeslání dat programem, jsou tato nejprve předána vrstvě, která doplní hlavičku požadovaného aplikačního protokolu, čímž vznikne paket. Pak jsou data předána transportní vrstvě. Po přidání hlavičky se datový celek označuje jako segment. Na síťové vrstvě se už jedná o paket (nebo datagram) a konečně na vrstvě síťového rozhraní mluvíme o rámcích. Při přijímání dat postupuje tento proces, v nejčastějších případech, opačně. Takže jak jsou při odesílání přidávány hlavičky, jsou při přijímání na jednotlivých vrstvách odstraňovány. Samotný program, který chce data přijmout či odeslat, má obvykle přístup pouze k hlavičce použitého aplikačního protokolu.

2.3 Protocol stack

Protože na každé vrstvě sady komunikačních protokolů (např. TCP/IP) může být použito více jednotlivých protokolů, má každá vlastní aplikační protějšek v jádře systému. Například pro protokoly TCP/IP je to IP modul. Ten je použit pro další zpracování datové jednotky ze síťového rozhraní, pokud nižší vrstva zjistí, že je použito protokolů z rodiny TCP/IP. Obecně řečeno, protocol stack obsahuje součásti systému, které definují způsob zacházení s daty určenými pro síťové rozhraní. A to nejen z hlediska přenosových protokolů ale také vzhledem k alokaci systémových prostředků. V literatuře se většinou se používá obecné označení „network stack“ nebo „TCP/IP stack“ podle nejpoužívanější architektury TCP/IP. V této práci bude protocol stack označován česky jako protokolový zásobník.

Pokud bude potřeba přistupovat k informacím z hlaviček na nižších vrstvách, než aplikační, bude nutné provést některé změny v rámci protokolového zásobníku i v případě použití knihovny. Což znamená, že softwarové součásti zásobníku mohou ovlivňovat funkci systému změnou některých nastavení, ale především snížením výkonu. Většinou se jedná o přístup více entit ke stejným systémovým prostředkům. To mohou být stejná místa v paměti, čas procesoru nebo třeba přístup do fronty. Konkrétně zde mohou způsobovat problémy opakovaná spojení, protože data jsou zpracovávána nejen při příchodu z vyšší vrstvy ale i pokud nejsou vyřízena ve frontě. Tato nevýhoda je ještě umocněna v případě bezdrátového rozhraní kde, jak vyplývá z podstaty, nelze vysílat a přijímat data současně.

V některých případech není vhodné využívat protokolový zásobník definovaný jádrem systému. Příkladem může být zbytečné plýtvání systémovými prostředky, pokud je třeba pracovat pouze se zlomkem dat. Protože pokud aplikace, či knihovna žádá informace jen z některých datových celků (paketů) na nižší vrstvě protokolového zásobníku, zpracují se většinou všechny datové celky.

To lze částečně řešit duplikováním některých funkcí zásobníku. Pro extrakci dat se využije obdobný subsystém, který je pro tyto účely přizpůsoben (např. knihovna). Tento přístup ovšem naráží na problém vícenásobného přístupu k výstupní frontě síťového rozhraní. Přesto je vhodnější než použití protokolového zásobníku, který využívá jádro systému. Je to dáno především tím, že většina prostředků pro zpracování dat z rozhraní nekopíruje veškeré funkce protokolového zásobníku jádra systému a soustřeďuje se jen na určitou oblast.

Pokud v systému běží monitorovací aplikace, je zachycený rámec duplikován pro účely jádra systému, různé síťové moduly a pro aplikaci. V jednu chvíli tak jsou zpracovávány dvě a více kopií jednoho rámce. Proto se protokolový zásobník při zachytávání paketů snaží o co nejefektivnější přístup.

Problém vícenásobného přístupu se různé systémy pro monitorování snaží řešit různými způsoby. Jde především o co nejjednodušší získání rámce od rozhraní a jeho následné efektivní zpracování. Nejlepším přístupem je zřejmě přímý přístup k paměti DMA. K tomu je využit ovladač, který zajišťuje alokaci a přidělování paměti pro síťové rozhraní. Ten při přijetí rámce informuje monitorovací aplikaci (respektive prostředky, které jsou použity pro získávání dat od rozhraní) pomocí přerušení nebo výzev polling.

Rámec je sice stále duplikován, ale omezuje se problém vícenásobného přístupu k výstupní frontě. V současné době existuje více řešení, které ale nemusí být vždy použitelné, především kvůli nutné implementaci do operačního systému.

2.4 Přístup ovladačů k paměti

Samotné síťové rozhraní nemá vlastní paměť. Její přidělení a alokace závisí na ovladačích síťového rozhraní a na jádru systému. Pro samotný přenos dat od síťového rozhraní ke vstupní frontě systému se používá několik metod. Ty mají rozdílné parametry nejen vzhledem k efektivitě práce s pamětí ale i v závislosti na vytížení procesoru. Protože si operace s pamětí žádají výpočetní čas procesoru, musí nějakým způsobem docházet k přerušení. Samotné přerušení spotřebuje nezanedbatelné množství systémových prostředků a je žádoucí jeho dopady na systém minimalizovat. Protože právě při operacích s pamětí dochází ke ztrátám rámců je, vhodné zvolit ovladač, který podporuje efektivnější přístup k systému.

2.4.1 Přerušení

Přerušení je jedním z mechanizmů, které zajišťuje využití procesoru více aplikacemi. Pokud aplikace, nebo třeba HW ovladač potřebuje využít procesor, zavolá přerušení a pokud splní podmínky je jim určitý čas na něm přidělen. Podmínky jsou dány obsluhou přerušení, která je závislá na službách jádra a jejich strukturách. Přerušení zajišťuje asynchronní přidělování systémových prostředků. [1]

2.4.2 Nevýhody přerušení

Pokud je protokolový zásobník informován o přijetí a uložení rámců do paměti přerušením, může při větším provozu dojít ke stavu, který se označuje jako bouře přerušení [3]. Pokud je přijato více malých rámců rychle po sobě, systém nestihne vyřídit první a zpracování je přerušeno. Fronta se tak plní nevyřízenými rámci, které samy generují přerušení, což vede k nečinnosti až selhání systému. Proto mají větší rámce za běžných podmínek v Linuxu menší ztrátovost než malé. Tento problém částečně řeší polling a obsluha přerušení. I když na nových systémových jádrech k problémům s přerušením téměř nedochází, bouře přerušení ilustruje nutnost správného řízení systémových prostředků

2.4.3 Metoda výzev (polling)

Metoda výzev slouží k rozdělování systémových prostředků obdobně jako přerušení. Má ale složitější strukturu, která zavádí stavy nebo události zařízení. Pokud aplikace potřebuje využít systémových prostředků, nevolá přerušení, ale čeká na výzvu. Pokud aplikace potřebuje provést více operací na stejném systémovém prostředku, může se jich provést více na základě jedné výzvy [1]. Pokud je například třeba předat aplikaci rámec z výstupní fronty rozhraní, může se jich aplikaci předat více na základě jedné výzvy.

2.4.4 DMA (Direct Memory Access)

Síťová rozhraní dnes nejčastěji používají, tzv. přímý přístup do paměti [7]. Ten minimalizuje účast procesoru na ukládání dat z rozhraní do paměti. V tomto případě se nejčastěji využívá asynchronní přístup a kruhový zásobník (ring buffer). V závislosti na rozhraní a jeho ovladačích může být zásobník umístěn v paměti sdílené s procesorem. Každý příchozí rámec je umístěn do jednotlivého zásobníku v kruhu a jeho zaplnění je signalizováno přerušení. Po přenesení určitého počtu rámců je vytvořen nový kruhový zásobník.

Jakmile síťové rozhraní přijme rámec, zavolá přerušení, aby informovalo operační systém o přijetí. Obslužná rutina přerušení alokuje paměť a předá informace rozhraní, které zde uloží data. Po uložení dat, je opět zavoláno přerušení a jeho obslužná rutina (handler) probudí odpovídající proces nebo vlákno. Při zpracování dat aplikací se obslužná rutina přerušení stará o režii.

2.4.5 NAPI (New API)

Zatímco DMA používá pro signalizaci přerušení, NAPI [8] využívá polling. Ten musí podporovat ovladače síťového rozhraní, také je nutné deaktivovat jejich přerušení. Pokud je síťovým rozhraním přijat rámec nezpracuje se ihned. Nejprve je zkontrolováno, jestli už nebylo voláno přerušení za stejným účelem. Tedy informovat protokolový zásobník jádra systému o nutnosti přijetí rámce. Pokud ještě nebyly obslouženy předchozí rámce a systémové prostředky se uvolní, jsou do paměti uloženy všechny přijaté rámce najednou. Pokud se příchozí rámce nestihnou obsloužit, jsou jednoduše přepsány ve frontě síťového rozhraní a nezatěžují tak jádro systému případnou dealokací. Tento přístup očividně čerpá více systémových prostředků a vyplatí se pouze, pokud je síťové rozhraní zatížené. Vzhledem k tomu, že zpracování dat z rozhraní je primární funkcí přístupového bodu, to ale znamená jen malou nevýhodu.

2.4.6 TNAPI (Threaded NAPI)

Tento mechanizmus nahrazuje přerušení a polling předchozích variant. Každá výstupní fronta rozhraní má přiděleno jedno vlákno kernelu, přičemž se může jednat i o virtuální rozhraní. Takže není nutné čekat, až se některé uvolní a zpracuje data od rozhraní. Také se usnadňuje přesun dat od protokolového zásobníku, na úrovni jádra systému, ke knihovně nebo programu na aplikační úrovni. Tento postup je výhodný především pokud je k dispozici procesor s více jádry, kdy se každé jádro stará o jedno vlákno kernelu. Bezdrátové přístupové body se dnes bohužel ve více jádrovém provedení nevyrábí. TNAPI lze ale i tak s výhodou využít, při monitorování virtuálních síťových rozhraní [4].

2.5 Vyhodnocení a zpracování dat

Jak již bylo napsáno v úvodu, informace získané analýzou provozu mohou sloužit širokému spektru aplikací. Účelem této práce je sledování zatížení přístupového bodu a vytvoření statistik. Pro tento záměr se bude využívat objem přenesených dat v určitém časovém intervalu, který se nejlépe hodí pro následné zpracování.

Objem provozu je dán službami, které jsou přes síť poskytovány. Služby nejsou poskytovány ihned, ale postupně podle použitého protokolu, navíc je poskytnutí vždy spojeno s určitou režií. Tyto faktory nejsou samozřejmě náhodné pouze neznámé. Jejich určení by bylo možné ale interpretace by vyžadovala implementaci určitých algoritmů. Konkrétně reverzních k těm, které používají přenosové protokoly. Jejich výsledek by ale měl na výsledku mnohem menší podíl než náhoda. V přístupové síti je provoz dán především chováním uživatelů, které může být popsáno například behaviorálními algoritmy. Rozsah a zaměření této problematiky jde ale daleko za hranice této práce. Proto bude pro zjednodušení předpokládáno, že na síťovém provozu se podílí pouze náhodné parametry. Je tedy možné jednotlivé vzorky provozu označit jako náhodnou veličinu?

2.5.1 Objem provozu jako náhodná veličina

Náhodná veličina je hodnota získaná z určitého systému podmínek, nebo pokusu. Opakovaným prováděním dochází v důsledku náhodných vlivů k jejím změnám. Množina všech těchto hodnot se nazývá základní soubor. V něm lze očekávat určité rozdělení pravděpodobnosti, například normální (Gaussovo). Náhodná veličina je diskrétní nebo spojitá podle toho zda lze hodnoty ze základního souboru určit pomocí přirozených čísel.

Předpokládá se, že objem provozu může nabývat hodnot od nuly do maximální kapacity přenosového kanálu. Ten závisí na použitém standardu, ale obecně lze říci, že objem dat nabývá hodnoty ze spočitatelné množiny. Tedy, že lze určit pomocí přirozených čísel. Také je jisté, že v určitém časovém intervalu bude přeneseno nenulové množství dat pro provozní potřeby. Díky tomu lze objem dat označit za diskrétní náhodnou veličinu.

I bez určování distribuční funkce lze předpokládat, že náhodnou veličinu půjde popsat pomocí statistického rozdělení pravděpodobnosti. Některé statistické parametry pak bude možno použít v metodách zpracování a odhadu vývoje dat.

2.5.2 Klouzavé průměry

Jednoduchý klouzavý průměr

Pokud je potřeba vyhodnocovat trend datové řady v reálném čase, bez určení konce intervalu, lze použít několik základních prostředků. Nejjednodušším je klouzavý průměr, v anglické literatuře označovaný jako Simple Moving Average (SMA). Ten je základním interpretačním prostředkem ve statistice. Vyhlazuje náhodné odchylky vzorků a je proto vhodný k dalšímu zpracování při vyhodnocování dat a odhadování budoucích trendů. V intervalech o zvolené velikosti se vypočítává průměr. Jednotlivé intervaly jsou vždy posunuty o jednu časovou hodnotu vpřed. Výsledkem klouzavého průměru je tedy součet souboru hodnot dělený velikostí intervalu.

$$I = \frac{1}{N} \cdot \left[V_n + V_{n-1} + V_{n-2} + \ldots + V_{n-(N-1)} \right] \tag{2.1}$$

Takovéto určení hodnoty z intervalu vnáší do výsledku určitou setrvačnost. Ten totiž závisí stejným způsobem na všech předchozích hodnotách. Proto se i významný nárůst na konci intervalu může projevit jen mírným zvýšením klouzavého průměru. Tento problém částečně řeší exponenciální klouzavý průměr.

Exponenciální klouzavý průměr

Obdobně jako jednoduchý klouzavý průměr funguje i exponenciální. Každá hodnota má ale jinou váhu, podle toho jak je vzdálena od právě vypočítávané hodnoty. Počítá se nejen s hodnotami dat z intervalu ale také s předchozí vypočítanou hodnotou. Té je zároveň přiřazena největší hodnota oproti ostatním, protože má k právě vypočítávané nejblíže.

Zpočátku je k dispozici řada vzorků V, každý v čase *n*. Pro stanovený časový intervalu je určen koeficient *α*. Ten slouží obdobně jako počet vzorků u SMA k určení průměru. Dále ale společně s koeficientem *k* = 0, 1, 2 **…** určuje váhu jednotlivých vzorků.

Váha je tím větší, čím blíže je jednotlivý vzorek ke konci časového intervalu, tedy k umístění v čase *n*, kdy se exponenciální průměr vypočítává. Vzorky o různých vahách zmenšené koeficientem *α* na hodnotu průměru tvoří větší část vzorku průměru *I*. Zbytek je předcházející hodnota *I*, která má sice přiřazenu nejmenší váhu ale je přičtena přímo.

$$I_n = \left[\alpha \cdot \sum_{k=0}^{k=n-1} (1-\alpha)^k \cdot V_{n-(k+1)} \right] + (1-\alpha)^{k+1} \cdot I_{n-(k+1)} \tag{2.2}$$

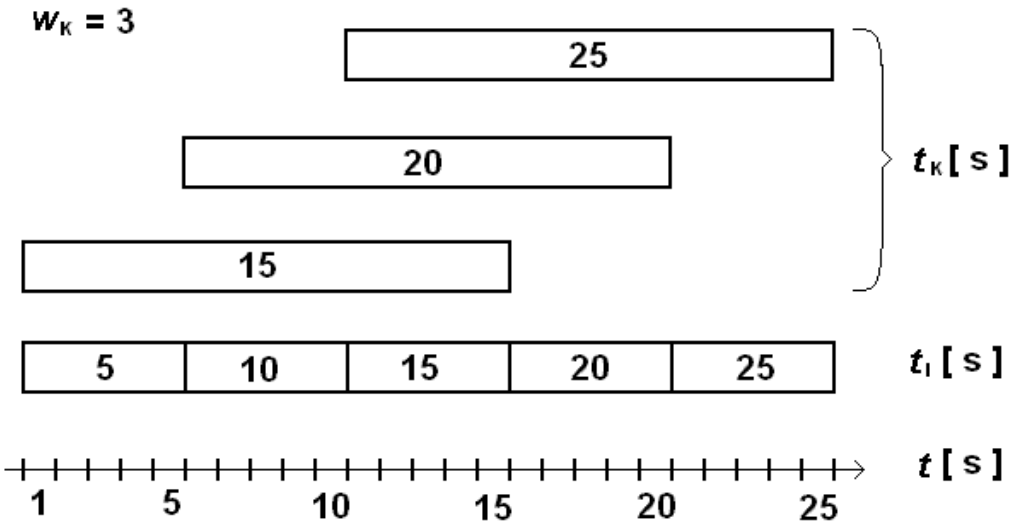
$$\alpha = \frac{2}{N-1} \tag{2.3}$$

2.5.3 Stochastický oscilátor

Stochastický oscilátor [15] se v technické analýze využívá pro signalizaci vzestupu a poklesu trendu. Využívá předpoklad, že zvýšení objemu dat bude po určité době následováno snížením se srovnatelným objemem. Obrat ve vývoji trendu je indikován, podle maximální a koncové hodnoty. Čím větší je mezi nimi rozdíl, tím větší je pravděpodobnost obratu. Výsledkem stochastického oscilátoru jsou indikátory v procentech. Pokud je hodnota blízko k minimu, je signalizován nárůst a pokud je blízko k maximu pokles. Zjednodušeně řečeno velmi prudký nárůst objemu dat signalizuje pokles a naopak. V praxi se volí jako krajní hodnoty 20% a 80%. Indikátor s hodnotou mezi nimi nesignalizuje nic.

Indikátor K

Základní jednotkou pro výpočet stochastického oscilátoru v technické analýze trhu je denní interval. Takovéto časové rozdělení je pro provoz nevhodné. Označíme ho proto jako základní interval a jeho délku *I* zvolíme jako libovolný počet sekund. Aby nedocházelo ke zkreslení, jsou všechny další časy závislé na čase *t* (kde *t = 0, 1, 2, 3 … n*)ve kterém byly zachyceny vzorky, které základní interval tvoří. Každý základní interval je označen časem, ve kterém byl zpracován. Tedy vždy v čase posledního zachyceného vzorku. S tímto časem je dále počítáno a proto je označen jako $t_I = t \times I$ (kde t_I = 0, 1**I*, 2**I*, 3**I* … *n***I*). V tomto souboru hodnot jsou již data vyhodnocována a uchovávají se pouze vybrané vzorky potřebné pro následné statistické zpracování.



Obr. 2.2: Základní intervaly a okno stochastického oscilátoru

Pro výpočet je dále nutné určit, v jak velkém časovém okně w_K budou hodnoty ze základních intervalů, zpracovávány do indikátoru *K*. Okno tedy určuje počet základních

intervalů, které se použijí jak je vidět na obr. 2.2. Ve výpočtu se ze všech základních intervalů v okně vybere maximální, minimální a koncový vzorek. (Vybírá se vždy pouze jeden vzorek ze všech dostupných základních intervalů.) Pro výpočet jsou použity vzorky v čase základních intervalů t_I. Pomineme-li počáteční stav, provádí se výpočet K v každém čase základního intervalu. Po dosazení do vzorce je výsledkem hodnota v procentech.

$$K_{t_I} = \frac{V_{end} - V_{\min}}{V_{\max} - V_{\min}} \cdot 100 \tag{2.4}$$

Pokud je třeba, určuje se K nikoliv pro jeden interval, ale pro w_K intervalů. V tom případě je čas t_I vynásoben w_K, vyberou se potřebné vzorky z takto určeného intervalu, a výpočet se provede opět podle vzorce 2.4. První indikátor K tedy lze určit až v čase $t_I * w_K$ kdy je k dispozici dostatečný počet základních intervalů pro naplnění časového okna w_K. Pro usnadnění výpočtu je zaveden celočíselný koeficient k**,** který se inkrementuje při každém výpočtu K. Koeficient je k základnímu času vztažen dle vzorce $k=t_I/I$.

$$t_K = (k - w_K + 1) \cdot i \tag{2.5}$$

Indikátor D

Protože samotný indikátor K je náchylný ke zvýraznění většiny nevýznamných odchylek od průměru, zpracovává se dále do indikátoru D. Ten představuje klouzavý průměr ze souboru hodnot K, přičemž se počítá s w_D hodnotami indikátoru K.

$$t_D = (k - w_D \cdot w_K + 1) \cdot i \tag{2.6}$$

$$D_{t_D} = \frac{\sum_{t_D-(w_D \cdot t_K)}^{t_D} K_{t_K}}{w_D} \tag{2.7}$$

Tyto dva základní indikátory K a D se označují jako Fast Stochastic a používají se dále k výpočtu dalších parametrů. Slow Stochastic se skládá opět ze dvou parametrů K_S a D_S kde K_S je D z Fast Stochastic. D_S je jako v předchozím případě klouzavý průměr z K_S, opět pro počet hodnot w_D. Tyto nové indikátory K_S a D_S jsou ještě méně citlivé na odchylky od trendu. Dnes se lze setkat také s indikátory Full Stochastic ale opět se jedná pouze o několikanásobné zprůměrování stávajících hodnot K ve větším okně. V podstatě jde vždy o to, zvolit vhodnou délku okna tak, aby indikátor co nejlépe charakterizoval vývoj trendu.

2.5.4 Indikátor Bollinger Bands

Bollingerovo pásmo[15] (dále jen BB) vytváří horní a dolní hranici okolo průměrné hodnoty. Tím je určen pravděpodobný výskyt většiny hodnot. Lze tak lépe odhadnout významné odchylky, které se dosud vyskytly. Toho se dá využít i k odhadu vývoje trendu, případně k určení chyby odhadu.

Vychází se ze statistické distribuce dat, konkrétně náhodné veličiny. V tomto případě se předpokládá pravděpodobnost výskytu hodnot podle normálního (Gaussova) rozdělení. To lze ve většině případů použít pro pravděpodobnostní rozdělení jak diskrétní, tak i spojité náhodné veličiny. Protože bylo na začátku stanoveno, že objem provozu bude brán jako diskrétní náhodná veličina, můžeme BB použít i pro účely této práce. Konkrétně se zde předpokládá, že 99,7% hodnot je v rozsahu třech standardních odchylek *σ*. Těch se pak využívá k určení BB.

Pro výpočet normálního rozdělení je nutné znát průměr a k němu současný rozptyl σ^2. Ten se určí sečtením mocnin rozdílů jednotlivých hodnot ze základního intervalu oproti průměrné hodnotě intervalu.

$$\sigma^2 = \sum_{i=1}^{k} (V_i - V_{avg})^2 \tag{2.8}$$

Z rozptylu se odmocněním určí standardní odchylka σ. Čím je odchylka větší, tím větší je pravděpodobnost výskytu velkých odchylek od průměru. Porovnáním s průměrem lze rovněž zjistit variační koeficient, který lze použít pro srovnání jednotlivých odchylek.

Hodnota standardní odchylky σ se pak již přímo používá pro vytvoření BB pásma prostým sečtením či odečtením se současnou hodnotou průměru. Tím vzniknou dvě křivky, které vymezují Bollingerovo pásmo.

$$BB_{up} = V_{avg} + k_o \cdot \sigma \tag{2.9 a}$$

$$BB_{down} = V_{avg} - k_o \cdot \sigma \tag{2.9b}$$

Počet standardních odchylek k_o, který se použije, závisí na lidském faktoru. Podle normálního (Gaussova) rozdělení platí:

- 68.27% hodnot leží mezi V_{avg} a V_{avg} ±1 x σ
- 95% hodnot leží mezi V_{avg} a V_{avg} ±2 x σ
- 99.7% hodnot leží mezi V_{avg} a V_{avg} ±3 x σ

Protože se výpočty provádí v reálném čase, je nutné stanovit velikost okna w_{BB}, z něhož se budou vybrat hodnoty pro výpočet. Toto okno je stejné jako interval pro výpočet

klouzavého průměru, aby nedocházelo ke zkreslení vlivem rozdílných výchozích hodnot. V BB je možné použít jak průměr intervalu tak jednoduchý nebo exponenciální klouzavý průměr. Je vhodné použít stejný typ průměru, který bude následně použit pro odhad.

2.5.5 Testování hypotéz

Protože dále v práci bude použita kombinace prostředků jak ze statistiky, tak z geometrie, nelze jednoduše převzít veškeré metody z testování statistických hypotéz nebo odhadů. Proto budou některé metody upraveny dle potřeb této práce. Protože původní záměr se vztahoval k odhadům vývoje provozu, byly realizovány některé nestandardní způsoby zpracování dat z technické analýzy. Ty budou použity pro tvorbu základního souboru. Odhady a některé jejich části lze použít i k testování samotných hypotéz.

Testování bude v tomto případě sloužit k určení vhodné metody pro odhad vývoje trendu. Jak již bylo napsáno v úvodu této kapitoly, objem přenesených dat bude rozdělen do časových intervalů, které budou brány jako jednorozměrný statistický soubor. Protože se bude odhadovat hodnota v následujícím časovém intervalu, půjde v podstatě o bodový parametrický odhad. Jeho přesnost ale nebude určována na základě směrodatné chyby. Protože se odhaduje budoucí bod v intervalu, bude chyba určena jednoduše na základě procentuální odchylky od skutečné hodnoty.

Testováno bude několik způsobů odhadu, hypotéz. Obdobně jako u odhadu, i zde se testují konkrétní hodnoty a nikoliv pravděpodobnostní rozdělení. Proto se jedná o parametrické testy. Zpočátku je třeba zvolit alternativní hypotézu, se kterou se budou ostatní porovnávat. Tím se zjistí, zda hypotéza vyhovuje či ne. Základem odhadu bude poslední hodnota klouzavého průměru, podle toho zvolíme alternativní hypotézu. Ta bude dána poslední hodnotou klouzavého průměru s tolerancí danou absolutní hodnotou standardní odchylky. Tím bude vymezena oblast přípustných hodnot. Pokud bude hypotéza produkovat hodnoty mimo tento interval, bude shledána jako neplatná. Jednotlivé hypotézy se budou provádět vícenásobně a dále srovnávat podle řady odchylek od skutečné hodnoty a podle celkové průměrné odchylky.

2.5.6 Extrapolace

Extrapolace je obdobou interpolace. Ta má za úkol nalézt funkci, která co nejlépe aproximuje hodnoty z určitého intervalu, v tomto případě základního souboru. Pokud jsou aproximovány všechny neznámé hodnoty ze všech hodnot základního souboru, označujeme interpolaci jako globální. Při použití jen některých okolních hodnot pro odhad jedné neznámé, mluvíme o interpolaci lokální. Dále se dělí podle funkce použité pro aproximaci.

Obecně platným způsobem je polynomická interpolace, kde se pro aproximaci hodnot používá mnohočlen stupně n-1, kde n je počet hodnot ve statistickém souboru. Celý statistický soubor je pak popsán jednou funkcí, ale může zde docházet k nepravděpodobným odchylkám. Nejjednodušším používaným způsobem je lineární interpolace. Pro dvourozměrný statistický soubor se tedy body prokládají přímkou. V celém základním souboru vzniká lomená přímka. Pak se předpokládá, že neznámé hodnoty v intervalu mezi dvěma body budou ležet v blízkosti této přímky. Při použití pouze dvou nejbližších okolních bodů v každé ose tedy platí:

$$y_n = y_{n+1} + (y_{n-1} - y_{n+1}) \frac{(x_n - x_{n+1})}{(x_{n-1} - x_{n+1})} \tag{2.10a}$$

kde x_n a y_n jsou souřadnice neznámého bodu.

Dnes se lze setkat i s dalšími funkcemi pro interpolaci, kdy se používají např. funkce vyšších řádů nebo splain křivky. Tyto způsoby jsou více výpočetně náročné a při náhodném rozdělení nezaručují lepší výsledky než lineární interpolace.

Extrapolace vychází ze stejných mechanizmů, pouze odhadované body leží vně statistického souboru. V praxi se používají pro odhady blízko koncové hodnotě intervalu, protože s rostoucí vzdáleností roste i možná chyba. V případě lineární extrapolace dvourozměrného souboru existují dva základní přístupy. První předpokládá konstantní průběh podle koncové hodnoty. Druhá počítá s tím, že neznámý bod bude ležet blízko přímky aproximující poslední (dva) body [16]. Pro statistický soubor o N prvcích, při použití dvou posledních bodů, platí:

$$y = y_N + (y_{N-1} - y_N) \cdot \frac{(x - x_N)}{(x_{N-1} - x_N)} \tag{2.10b}$$

kde *x* a *y* jsou souřadnice neznámého bodu.

3 VÝBĚR HARDWARE

3.1 Modifikovatelné přístupové body

V současné době je na trhu dostupná široká škála bezdrátových přístupových bodů. Na první pohled je lze rozdělit na hardwarově modifikovatelné a nemodifikovatelné. První skupina nabízí velice široké možnosti využití, nejen jako přístupový bod. Patří sem například přístroje firmy Mikrotik, Ubiquiti Networks, Alfa Networks, Alix atd. Pro tuto práci jsou však důležité možnosti rozšíření pouze z hlediska funkcí síťových rozhraní.

U modifikovatelných zařízení je tedy možno dodatečně přidat do zařízení více síťových karet. Běžné je použití dvou slotů pro karty v rozdílných pásmech, ale existují i přístroje např. se čtyřmi sloty. Pro dnes běžně používané bezdrátové standardy jsou ale vyhrazena jen dvě pásma. Mohlo by se tak zdát, že fyzická přítomnost více síťových adaptérů, není nutná.

Pro síťová rozhraní není samozřejmé, poskytovat data více aplikacím, většinou jsou předána pouze jádru systému. Přístupový bod s nimi ale musí pracovat a jejich sdílení bude pravděpodobně znamenat omezení výkonu. To na první pohled představuje výhodu pro modifikovatelné přístroje s více síťovými adaptéry. Pak je ale nutné na druhém adaptéru nastavit promiskuitní nebo monitorovací režim. Jedná se o nestandardní způsob využití, i když druhé síťové rozhraní se může ukázat jako zajímavá alternativa.

Dalším hlediskem je možnost vývoje vlastních aplikací pro daný firmware. Většina výrobců tuto možnost neposkytuje. Pouze někteří výrobci modifikovatelných přístupových bodů nabízí tuto možnost (Mikrotik). A jen výjimečně je poskytnuto vývojové prostředí pro aplikace (Ubiquiti Networks). V takovém případě je navíc komunita vývojářů velmi malá. Hrozí tak možnost, že při řešení dané problematiky nebude dostupná potřebná dokumentace nebo API. Otázka firmware bude podrobněji řešena později, zatím lze říci, že vývoj aplikací pro komerční modifikovatelné přístupové body nemusí přinášet očekávané výhody. Především v oblasti podpory od výrobce.

Nicméně modifikovatelné přístroje jsou určeny pro profesionální použití a jsou proto oproti nemodifikovatelným dostupné s výkonnějším HW, samozřejmě za odpovídající cenu. Rychlejší procesor, je pro vývoj aplikace jednoznačné pozitivum. Od procesoru se odvíjí výkonnost aplikace nebo třeba menší nutná míra optimalizace v závislosti na pokročilejší architektuře. Pozitivem je také možnost rozšíření o výkonnější síťový adaptér, nebo adaptér s vhodnějšími ovladači.

3.1.1 Promiskuitní a monitorovací režim

Pokud by bylo rozhodnuto, využít druhé síťové rozhraní ve stejném pásmu pro monitorování, muselo by toto využívat promiskuitní nebo monitorovací režim k zaznamenávání paketů.

Síťové rozhraní v běžném provozu přijímá všechny rámce v síti, ale do jádra systému propouští jen ty s MAC adresou systému. Pokud je v promiskuitním režimu propouští všechny rámce do jádra systému. To může znamenat zbytečnou zátěž síťového rozhraní i jádra systému, nejen při přijímání náhodných rámců. Systém se chová ke všem zprávám jako by byli z jeho sítě a tak může samovolně odpovídat na všesměrové zprávy, žádající o překlad IP na MAC adresu. Nebo žádat DNS server o překlad adres které nejsou z jeho sítě. Tyto nedokonalosti promiskuitního režimu se používají k jeho detekci. Síťové rozhraní tak může být mylně detekováno jako útočník a odpojeno ze sítě. Promiskuitní režim se používá v Ethernetu, i bezdrátových sítích. Rozhraní zde musí mít platnou síťovou adresu.

Monitorovací režim se používá pouze pro bezdrátová síťová rozhraní. Na rozdíl od promiskuitního může být pouze pasivní a rámce pouze přijímat. Ovladače některých výrobců dokonce využívají pouze vstupní frontu rozhraní. Tam kde se využívají obě fronty, může docházet k chybám jako neodstranění duplicity nebo neodeslání ACK paketu. Na rozdíl od promiskuitního režimu, nemusí mít stanice platnou síťovou adresu. Jednoduše zachytává všechny rámce v dosahu, které jsou vysílány na odposlouchávaném kanálu. Protože je třeba monitorovat pouze klienty v síti, jedná se vlastně o nevýhodu.

Pokud osadíme zařízení více síťovými rozhraními, je nutné počítat s tím, že si jednotlivá rozhraní budou konkurovat v přístupu k procesoru. Každou datovou jednotkou, která projde jádrem systému, se tak bude procesor zabývat dvakrát. Poprvé při vyřízení v režimu přístupového bodu a podruhé při zpracování pro aplikaci analyzující provoz. Obdobná situace sice nastává i při běžném monitorování jednoho rozhraní, ale zde není potřeba obsluhovat dvě fyzicky přítomná síťová rozhraní. Použití druhého rozhraní může při správném nastavení zmenšit ztrátovost zachycených paketů, ovšem za cenu mnohem vyšší míry optimalizace aplikace. Takovéto řešení, se proto v budoucnu pravděpodobně ukáže jako nevyhovující.

3.2 Dvoupásmové přístupové body

Jak již bylo zmíněno, pro některé způsoby použití přístupových bodů jsou vhodná dvě síťová rozhraní pracující v rozdílných pásmech. Dvoupásmové přístupové body umožňují vytvářet složitější topologie, studovat rozdíly mezi standardy v obou pásmech nebo vytvářet modifikace firmware či programy pro více síťových adaptérů atd.

Program, jenž bude součástí této práce, má využívat přístup k softwarovým komponentám. Je tedy třeba rozhodnout, zda se rozdílné standardy, nebo počet adaptérů neprojeví při návrhu aplikace. Proto byla prostudována dostupná dokumentace ovladačů s následujícím závěrem. Při monitorování sítě, nebudou rozdíly mezi standardy protokolu 802.11 patrné. To ani v případě programu, který by pracoval přímo s HW ovladači, protože specifika jsou definována na nižší vrstvě.

Navíc s největší pravděpodobností bude při návrhu použita softwarová komponenta na nebo nad úrovní kernelu. Nebude tedy přistupovat k HW přímo, ale bude využívat stejnou definici rozhraní jako jádro linuxu. Případná nutnost monitorování dalšího síťového rozhraní tedy půjde jednoduše naprogramovat i bez fyzické přítomnosti rozhraní. Dvoupásmové zařízení by bylo vhodnější, pouze pokud by bylo nutné vytvořit složitější topologii, která by využívala obou pásem. Je ale třeba počítat s tím, že obsluha dvou síťových rozhraní bude znamenat větší zátěž procesoru.

3.3 Přístup k firmwaru

Pro ovládání funkcí přístupového bodu je dnes používáno webové rozhraní, které běžně umožňuje přehrávání firmwaru. Nikoliv ale manipulaci s vlastní flash pamětí, kde je uložena softwarová výbava. Tento přístup probíhá běžným přímým ethernetovým kabelem připojeným zpravidla do jediného portu, který má přístup k procesoru a paměti. V tomto případě je nutné mít na přístupovém bodu funkční software. Přístup přes bezdrátové rozhraní není většinou možný. Chyba přenosu způsobená rušením, by mohla mít fatální důsledky např. při modifikaci firmware. Někteří výrobci používají COM port se specifikací RS-232, což je pravděpodobně nejbezpečnější přístup. Navíc je umožňuje přímý přístup k programovatelným prvkům, a lze proto použít k obnově při kritickém selhání softwaru. Pokud dojde k chybě na zařízení bez COM portu a nelze se dostat do failsafe[1] režimu zbývá přehrát paměť některou metodou pro přehrávání flash pamětí. Takovou metodou je např.

[1] Režim kdy je načtena pouze část paměti se základním nastavením

JTAG[2] nebo I2C. Zde už je potřeba nestandardních prostředků, kterými jsou většinou různě upravené sériové kabely s TTL převodníkem. Ani pak ale není úspěšné přehrání firmwaru zaručeno. Proto by zvolené zařízení mělo mít COM port.

3.4 HW parametry

Výkonnost přístupových bodů je dána především použitým procesorem a velikostí paměti. Zároveň je třeba brát v úvahu možnost použití otevřeného software, místo softwaru výrobce. Otevřený software je většinou postaven na upraveném jádře Linuxu. To umožňuje určit minimální požadavky na HW tak, aby běh software měl minimální vliv na výkonnost přístroje. Při studování zkušeností vývojářů i uživatelů s různými druhy otevřeného software byly zjištěny vždy velmi podobné požadavky na HW.

3.4.1 Operační paměť

Jako naprosté minimum pro běh jádra linuxu se uvádí 8 MB RAM. V tomto případě je přístroj schopen stabilně obsluhovat jen několik klientů navíc s omezenou funkčností. Proto se běžně užívá 16MB operační paměti. Pokud jsou potřeba pokročilejší funkce, mezi které patří zamýšlené monitorování provozu, je doporučená velikost operační paměti větší. Vzhledem k dostupným velikostem na trhu tedy 32 MB a vyšší.

Důležitou vlastností paměti je i její rychlost, respektive rychlost jakou s ní komunikuje procesor. Příchozí data od rozhraní se totiž před zpracováním musí ukládat do paměti. Možnosti obsluhy zařízení systémem jsou tedy omezeny rychlostí práce s pamětí. V současné době je toto negativum kompenzováno použitím vyrovnávací paměti procesoru a různými metodami současného přístupu k paměti.

3.4.2 Procesor

Většina výrobců bezdrátových přístupových bodů používá procesory s redukovanou instrukční sadou RISC nebo obdobu CISC, jenž používá Intel. RISC se vyznačuje jednoduchými instrukcemi, kdy se v každém strojovém cyklu provede alespoň jedna. Od této architektury se odvozují další. Nejpoužívanější jsou MIPS, ARM, PowerPC atd. Pro monitorování provozu je důležitá skutečnost, že jejich architektura a běžně používané ovladače umožňují předání informací od síťového rozhraní pouze jednomu subsystému v jednu chvíli. Dnes už existují řešení [4] předávající data z výstupní fronty více vláknům. Mají ale smysl především pokud

[2] Standard pro komunikaci s integrovanými obvody

má zařízení více jádrový procesor. Což v současné době nesplňují ani nejdražší varianty bezdrátových přístupových bodů.

Velká část levnějších zařízení na trhu využívá platformy Broadcom nebo Atheros. Ty jsou většinou postaveny na architektuře MIPS nebo MIPSel. Hlavní odlišností je poloha nejvyššího a nejnižšího bytu. Přičemž MIPSel využívá LSB, tedy nejméně významný bit je na nejnižší adrese v paměti. Ústřední myšlenkou architektury MIPS je možnost programátora částečně organizovat zpracování více instrukcí v procesoru najednou. To je samozřejmě možné pouze pokud instrukce nepotřebují využít stejné části procesoru. Pro vývoj aplikací se ale tato vlastnost většinou nepoužívá. O přizpůsobení výsledného binárního souboru programu pro procesor se téměř ve všech směrech postará vývojové prostředí nebo utilita použitá pro kompilaci.

Broadcom využívá procesory s rozšířenou instrukční sadou která je vhodná pro multimediální aplikace. Pro práci s pamětí ale používá dvě cache o velikosti pouze 16 KB. To může být při práci s rozhraním limitující. Proto používá hardwarovou jednotku pro správu paměti, která podporuje více vláknové aplikace.

Atheros má pro práci s pamětí vyhrazeny také dvě cache ovšem o velikosti 32 KB. Navíc může k paměti přistupovat až pět zařízení najednou. Jedná se o procesor, USB port, dva ethernetové porty a PCI port. Ten je většinou používán ke komunikaci s čipsetem bezdrátového rozhraní [13].

Dalšími zástupci na trhu jsou např. procesory Intel nebo PPC či ARM od různých výrobců. Ti využívají vlastní architekturu a používají se spíše v komplexnějších zařízeních, než jsou bezdrátové přístupové body. Jako nejvhodnější alternativa pro požadavky v zadání diplomové práce se tedy jeví procesor Atheros.

3.5 Ovladače síťového rozhraní

Atheros

Atheros již delší dobu používá architekturu, která spojuje funkci procesoru a síťové karty do jedné čipové sady. To umožnilo nižší výrobní náklady a zjednodušení návrhu zařízení. Zpracování příchozích signálů ale zřejmě probíhá odlišně než u klasické koncepce v PC zejména z hlediska plánování (přidělování systémových prostředků).

Ovládání HW pomocí ovladačů se běžně provádí prostřednictvím volání HW funkcí. S narůstajícím počtem rozdílných čipů a přístupů nebyli vývojáři ovladačů schopni zajistit kompatibilitu pro všechny. Proto se pro komunikaci jádra systému s HW začala používat abstraktní vrstva HAL (madwifi[3]) vyvíjený přímo Atherosem. Ten zprostředkovává API umožňující přístup k HW. Systémové prostředky jsou tak zbytečně zabrány přítomností další vrstvy při přístupu k rozhraní.

I když se dnes u Atherosu používají i jiné architektury, některé ovladače stále používají abstraktní vrstvu HAL. To sice usnadňuje vývoj, ale nepřímý přístup může mít za následek snížení výkonu nebo nestabilitu. Právě nízký výkon a nestabilita byli v minulosti Atherosu vyčítány. Novější architektury pro přístupové body už využívají běžné uspořádání dvou čipů, procesoru a NIC spojených sběrnicí. Rovněž ovladače z balíčků ath5k a ath9k již HAL nepoužívají a HW funkce jsou volány přímo. Při volbě HW je tedy vhodné zvolit architekturu využívající nové ovladače. Ovladače pro síťová rozhraní Atheros byly vytvořeny na základě softwarové struktury mac80211 která se vyvinula z vrstvy HAL. Mac80211 mimo jiné zprostředkovává virtuální rozhraní, které umožňuje vytvořit více síťových rozhraní na jednom fyzickém adaptéru. Díky tomu lze jedno virtuální rozhraní provozovat v režimu přístupového bodu a další v monitorovacím režimu.

Broadcom

Linuxové ovladače firmy Broadcom pro bezdrátová rozhraní dříve používaly zastaralou softwarovou strukturu softmac. Ta v poslední době přestala dostačovat novým standardům jako je IEEE 802.11/n a navíc nepodporovala šifrování. Proto jsou nové ovladače od verze 4 vytvořeny na základě struktury mac80211. Lze tedy očekávat, že ovladače firmy Atheros i Broadcom budou mít stejné vlastnosti. Rozdíly je tedy třeba hledat v odlišných procesorech a architekturách. Ty jsou ale teoreticky těžko srovnatelné, proto by bylo vhodné je porovnat v praxi.

[3] Název ovladačů síťového rozhraní a zároveň projektu, který rozšiřoval okruh podporovaných rozhraní.

4 VÝBĚR SOFTWAROVÝCH PROSTŘEDKŮ

Pro monitorování siťového provozu se dříve používala jednoúčelová zařízení a provádělo se spíše jen na důležitých síťových spojích. Rozvoj hardware umožnil rozšířit o funkci monitorování běžně užívané zařízení, jako jsou bezdrátové přístupové body. Protože se nejedná o běžně využívanou funkci, je třeba vhodně zvolit softwarové prostředky.

4.1 Volba firmware

Na funkce a služby poskytované přístupovými body jsou kladeny čím dál vyšší nároky. Zároveň stále stoupá výkon a klesá cena používaných HW prostředků. Především tyto dva faktory vedly většinu výrobců k vývoji firmwaru na základě jádra Linuxu. Umožňuje to mnohem lepší přenositelnost vytvořených aplikací a rozšiřuje tím jejich možnosti. Linux je pro implementaci síťových služeb vhodný především kvůli možnosti měnit mechanizmy jádra systému a uzpůsobit je pro cílovou aplikaci.

I když se jedná o otevřenou platformu, téměř žádní výrobci přístupových bodů neumožňují programově rozšiřovat či měnit firmware. Pouze někteří umožňují vývoj aplikací nebo API pro svou platformu. Také je třeba brát v úvahu, že vhodný firmware nemusí být dostupný pro vyhovující HW platformu a naopak.

4.1.1 RouterOS

Firmware dodávaný k výrobkům firmy Mikrotik je založen na Linuxu a má uzavřený zdrojový kód. Co se týče monitorování provozu má široké aplikační možnosti, které ale nelze plně využít pro vývoj aplikací. RouterOS je dostupný v několika verzích které se od sebe liší funkčností a cenou. Od verze tři umožňuje vytvářet vlastní aplikace. Ty je možné psát téměř v jakémkoliv jazyce, ale podléhají omezením, které vyplívá z API firmwaru RouterOS. Především aplikace se před spuštěním nahrává z připojeného PC. Přestože se veškeré instrukce provedou na přístupovém bodu, jedná se o nestandardní řešení. Také zápis programu do paměti na přístroji je omezen, což ztěžuje jeho vývoj. RouterOS není vyvíjen pro jiné HW platformy než Mikrotik, podpora jiných zařízení je tak problematická.

4.1.2 AirOS

Obdobně jako RouterOS je vyvíjen pouze pro jednoho výrobce HW a to pro Ubiquity Networks, konkrétně pro výrobky RouterStation. AirOS je k dispozici i s prostředky pro vývojáře, takže pro něj lze vytvářet aplikace přímo např. v jazyce C. Bohužel je komunita vývojářů malá a tak je k dispozici jen málo dokumentace a přizpůsobených knihoven.

4.1.3 SlugOS

Komerční firmware vyvinutý pro jeden typ přístupových bodů firmy Linksys s možností připojení externí paměti. SlugOS běží v paměti přístroje na rozdíl od předchozí verze Unslug. Jedná se, původně o na linuxu založený firmware uvolněný pro vývoj firmou Linksys. Protože zařízení se vyznačovalo především možností připojení externí paměti, byl systém uzpůsobený pro poskytování služeb související se síťovým datovým úložištěm jako souborový, mediální nebo webový server s databází. Proto SlugOS neoplývá příliš širokým spektrem aplikací, co se monitorování provozu týče. Také dokumentace je v tomto ohledu omezená. Pro SlugOS je možné vytvářet aplikace pomocí přizpůsobeného souboru utilit Toolchain, ale nabídka přímo podporovaných API nebo knihoven je poměrně úzká. Společně se špatnou dostupností HW, pro který je určený to znamená, že pro účely monitorování není příliš vhodný.

4.1.4 Oleg firmware

Jedná se o nezávislý firmware šířený pod veřejnou licencí. Lze ho tedy libovolně modifikovat a nástroje pro vývoj vlastních aplikací jsou rovněž volně k dispozici. Nabízí široký výběr knihoven a existujících aplikací. Bohužel podporuje jen některé konkrétní typy zařízení, což platí i o knihovnách. Může se tak stát, že potřebná knihovna bude dostupná, ale bude určena pro jinou architekturu (typicky MIPS a MIPSel). Knihovnu je pak nutné vhodně překompilovat, aby byla kompatibilní.

4.1.5 OpenWRT

Obdobně jako Oleg má otevřený zdrojový kód a je kompatibilní s mnoha aplikacemi a knihovnami. Databáze podporovaných přístupových bodů je velmi široká a téměř nezávislá na výrobci nebo použité čipové sadě. Dokumentace stejně jako knihovny jsou dostupné pro každý přístroj, většinou i s funkční a odzkoušenou verzí systému. Ta je často ve formátu, který používá výrobce, takže jí lze nahrát přímo z menu původního firmware [11].

OpenWRT se neustále vyvíjí, což je dobře zdokumentováno jak v případě vylepšení, tak v případě chyb. Komunita uživatelů a vývojářů je velká a tak lze očekávat, že případné problémy při vývoji či implementaci aplikací budou rovněž vyřešeny a zdokumentovány.

4.2 Dostupné programovací jazyky

Nejsnazší cestou k vytvoření aplikace je použití skriptu nebo skriptovacího jazyka. Ty často umožňují čerpat ze systému komplexní informace nebo snadno provádět změny nastavení. Pro složitější aplikace nejsou příliš vhodné, protože čerpají mnohdy nezanedbatelné množství systémových prostředků. To je problematické především na přístupových bodech, které nedosahují výkonů dnes běžně používaných PC. Z obdobného důvodu není vhodná většina interpretovaných [4]jazyků. Proto je nutné použít programovacích jazyků jako je C, C++ nebo Java

Volba jazyka je podmíněna několika předpoklady. Především jde o podporu na cílovém systému, který musí poskytovat vývojové prostředí a vhodné API či knihovny pro tvorbu aplikace. Dále jsou důležité možnosti přizpůsobení protokolového zásobníku, které umožňují některé knihovny. Efektivita se projeví při přístupu k datům síťovému rozhraní, jejich zpracování a efektivitě předání aplikaci. Na to má vliv složení a nastavitelnost samotného protokolového zásobníku a metody použité při žádosti o přidělení systémových prostředků. V literatuře se většinou jako významný parametr při efektivním monitorování uvádí také zachytávaná délka[2]. Ta představuje množství dat, se kterým je nutné manipulovat, než se požadovaná data dostanou od fronty síťového rozhraní k aplikaci.

4.2.1 Java

Programy napsané v jazyce Java využívají pro přístup k hardware JRE, což je především virtuální stroj JVM a soubor standardních knihoven. Při každém spuštění programu běží i odpovídající JVM. To vyvolává předpoklad, že bude oproti jiným jazykům méně efektivní při čerpání systémových prostředků. Díky neustálému vývoji a přizpůsobení JVM jádru systému je tomu často naopak, protože optimalizaci z velké části nemusí řešit programátor. To je samozřejmě efektivnější než využití samotného jádra systému, které nelze optimalizovat jen pro jednu aplikaci.

Použití Javy by bylo výhodné hned z několika důvodů. Jednak kvůli zmiňované menší nutné míře optimalizace aplikace programátorem. Dále kvůli sadě knihoven i samotnému programu, který nemusí být přizpůsoben kompilátorem pro cílový systém, ale pro pouze JVM. Pozitivem jsou i široké možnosti nastavení samotného JVM, která mohou zmenšit ovlivnění systému na minimum.

[4] Nepřekládají se přímo do strojového kódu, ale až za běhu pomocí softwarového interpreteru

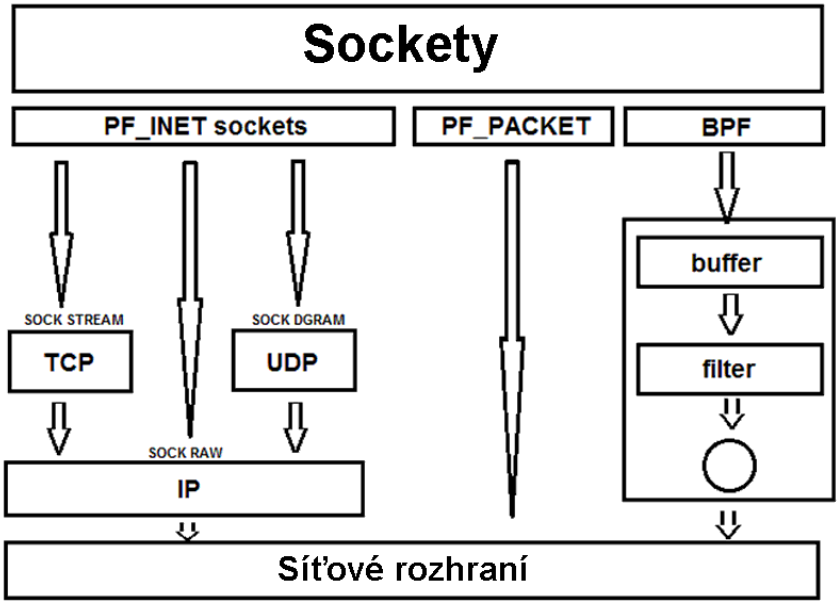
Využití JRE umožňuje velmi dobrou přenositelnost programů mezi operačními systémy a efektivní přístup k jádru systému. JRE ale musí být pro daný operační systém přizpůsoben. To může být problém v případě, že se ho chystáme použít na firmware s upraveným jádrem systému. V současné době bohužel není u zvoleného firmware k dispozici stabilní verze JRE.

4.2.2 Jazyk C/C++

Většina současných programů pro monitorování sítě je napsána v jazyce C v kombinaci s knihovnou pcap, ať už se jedná o Linux nebo Windows. Je to především z historických důvodů, protože jazyk C byl vyvinut pro psaní systémových aplikací, ovladačů atd. Také jádro Linuxu je z větší části napsáno v tomto jazyce. To zaručuje kompatibilitu v případě práce se systémem, i když v tomto ohledu bude v aplikaci využito funkcí knihovny. Na té také závisí efektivita zpracování dat od rozhraní na úrovni systému. Vzhledem k jejímu použití v profesionálních monitorovacích nástrojích se jedná o ověřenou variantu. Co se týče kompatibility s firmware, pokud daná verze umožňuje vývoj aplikací je vývojové prostředí pro jazyk C základním předpokladem.

4.3 Volba prostředků

Pro monitorování síťového provozu lze použít dvou základních knihoven. První je `socket.h` která slouží především k síťové komunikaci mezi dvěma aplikacemi [5]. V určité konfiguraci ale může sloužit i k zachytávání provozu. Druhou je `libpcap.h` která je přímo pro zachytávání určena [10]. Knihovna zajišťuje na implementaci nezávislé programové rozhraní pro přístup k prostředkům jádra systému. Pomocí systémových funkcí a procedur umožňuje ovlivňovat data síťového rozhraní na nižších vrstvách modelu použitého komunikačního protokolu. Aby bylo možné porovnat rozdíly mezi knihovnami je nutné vysvětlení několika pojmů, které se týkají zpracování dat síťového rozhraní na úrovni jádra systému.



Obr. 4.1: Srovnání přístupu k rozhraní pro domény socketů a BPF (BSD Packet Filter)

4.3.1 Sockety

Při návrhu aplikací využívajících síťové rozhraní se dnes v Linuxu používá především Berkley socket [5]. Ten umožňuje komunikaci dvou aplikací, i když běží na rozdílných strojích v síti. Je to dáno použitím sady komunikačních protokolů, které řeší specifika přenosu. Možnosti socketů jsou tak definovány především použitou sadou protokolů. Samotná data, která jsou předávána rozhraní, obsahují pouze hlavičku aplikačního protokolu. Ty ale neobsahují typ informací potřebných pro tvorbu statistik síťového provozu. Například pro monitorování provozu jednotlivých uživatelů, je důležité zachytávat všechny data, bez ohledu na použité aplikační protokoly. Proto je nutný jiný přístup a to na základě IP adresy. Takový umožňují typy socketů RAW_SOCK. Pokud jsou použity v rodině protokolů AF_INET (PF_INET) umožňují číst informace z transportní a síťové vrstvy. Je to dáno tím, že

protokolový zásobník jádra systému neprovádí odpouzdření datové jednotky, na síťové a transportní vrstvě, ale předá ji v surové formě aplikaci. Protože aplikace bude běžet na přístupovém bodu, který má poskytovat určité služby. Například pro směrování jsou třeba data z hlaviček protokolu síťové vrstvy. V tomto případě by tedy aplikace měla zastoupit určité funkce protokolového zásobníku jádra systému na síťové a transportní vrstvě. Provozovat veškeré funkce operačního systému ale není cílem této práce. Je tedy třeba jiný způsob, který bude duplikovat funkci protokolového zásobníku a zachová funkci jádra systému.

Pro příjem dat z vrstvy síťového rozhraní byl původně určen typ socketu SOCK_PACKET. Zde se jednotlivé rámce se neukládají do zásobníku, ale jsou předávány jednotlivě. Socketu jsou navíc předávány veškeré rámce podle zvoleného protokolu vrstvy síťového rozhraní a nelze definovat žádný filtr. To způsobuje vysokou zátěž systému a tento způsob není pro monitorování vhodný.

1) PF_SOCKET

Další možnost monitorování síťového rozhraní nabízí kombinace RAW_SOCK s rodinou PF_PACKET [5]. Ta přistupuje přímo k datům síťového rozhraní, navíc využívá upravenou verzi protokolového zásobníku jádra systému, která není tak robustní jako předdefinovaný zásobník jádra systému. Zároveň je možné omezit informace, které jsou extrahovány z hlaviček rámců, segmentů a paketů. Tím se omezí zachytávaná délka (capture lenght). Tyto dvě vlastnosti slibují větší efektivitu zpracování dat od síťového rozhraní.

Pro zachytávání lze použít téměř běžné socketové programování. Při tvorbě socketu se jako první atribut specifikuje rodina protokolů (doména) PF_PACKET s typem SOCK_RAW. Pak je nutné specifikovat standard příchozích rámců, dnes většinou Ethernet. Pokud chceme třídit pakety podle IP adresy, použijeme tedy protokol `ETH_P_IP` . To umožňuje přijímat veškeré pakety např. prostřednictvím funkce `recv()`.

2) PF_RING

Je obdoba domény PF_PACKET. Také přistupuje přímo k síťovému rozhraní a používá vlastní protokolový zásobník, který při použití vhodných ovladačů může kopírovat data přímo do vlastní kruhové fronty. Pokud běží více vláken se sockety domény PF_RING, vytváří si každé vlastní frontu. Do nich pak mohou být data z rozhraní předávána současně. Pro ušetření systémových prostředků se jednou přidělené fronty recyklují. Což znamená, že není alokována nová paměť, ale přepíše se obsah stávající. Způsob jakým toto předání probíhá, závisí na metodě přístupu k paměti, použité ovladači rozhraní.

4.3.2 BPF (BSD Packet Filter)

BPF [9] je původně vyvinut pro unixový operační systém FreeBSD, ale lze použít i v Linuxu společně s knihovnou pcap. BPF se vyznačuje použitím filtrů na nižších vrstvách protokolového zásobníku. Ty mají co nejméně snížit zátěž duplicitního zásobníku propouštěním jen žádoucích paketů. Například tak, aby aplikace obdržela pouze pakety s určitou cílovou adresou nebo portem. To se jeví jako velmi výhodné v případě, že je třeba sledovat jen některé klienty ze sítě.

V linuxových protokolových zásobnících dochází, ihned po zachycení každého rámce, k volání přerušení, nebo pollingu, pro uložení do paměti. U BPF je možné odchytit více rámců najednou a uložit je až následně. Přístup k paměti je možný dvěma způsoby, ale děje se vždy na základě dvou zásobníků. Ve výhodnějším (Zero Buffer) má aplikace načteny dva zásobníky. Jeden je naplňován síťovým rozhraním a z druhého jsou zpracovávána data aplikací. Po skončení zpracování je druhý zásobník zaměněn za první, který je naplněn daty od rozhraní. Po záměně dojde k přepsání starých zpracovaných dat novými a není tak zatěžovat systém alokováním nové paměť [2].

Pro každý použitý filtr se vytváří na registrech závislé pseudo zařízení, které pro definici filtru využívá vlastní jazyk. Filtr čte rámce přímo a nastavení filtru tak může být naprosto libovolné podle informací v hlavičkách. To také znamená protokolovou nezávislost. Rovněž je možné určit, kolik bitů z každé hlavičky bude předáno aplikaci. Data ze zásobníku se aplikaci předají až po stanoveném intervalu nebo po zaplnění fronty. Zpracování je tedy velmi efektivní především pokud aplikace žádá o malé množství informací z každého rámce.

5 REALIZACE

5.1 Instalace OpenWRT

OpenWRT se neustále vyvíjí, jak z hlediska funkcí tak vzhledem k hardware. Protože možnosti přístupových bodů neustále rostou, mění se i požadavky na software. Jakmile jsou možnosti jedné generace přístrojů vyčerpány, je ukončen i vývoj verze. Proto je v současné době k dispozici již třetí verze s názvem Backfire. Většina v současné době dostupných přístrojů je podporována právě touto verzí, některé starší přístupové body jsou kompatibilní s předchozí verzí Kamikaze.

Zdrojové kódy společně s nástroji pro vývoj a implementaci lze stáhnout z webového projektu správy OpenSource software s názvem Subversion. Zde se nachází vždy poslední stabilní verze se záplatami a aktualizacemi.

5.1.1 Kompilace jádra systému

Kompilaci firmware lze provést pouze v operačním systému Linux. Ten musí splňovat určité předpoklady, kterým v rozdílných distribucích odpovídají různé balíky nástrojů a knihoven. Nejlépe zdokumentované [11] jsou v tomto ohledu v současnosti Debian 5 a Fedora 11.

Volby kompilace jádra OpenWRT jsou tak rozsáhlé, že jsou uloženy v konfiguračním souboru a je možné je měnit v menu vyvolaném příkazem `make menuconfig`. Především je nutné zvolit cílové zařízení a to vzhledem k procesoru a použité síťové kartě. Dále je nutné vytvořit obraz systému a tedy zvolit cílový souborový systém. Pokud je zvolen jffs2 je nutné znát velikosti bloků (sektorů) cílové flash paměti.

Pro vývoj aplikací jsou důležité dvě volby, vytvoření SDK[5] a přizpůsobeného souboru utilit toolchain. SDK obsahuje zvolené knihovny a poskytuje prostředí pro tvorbu balíčků a aplikací na základě nástroje GNU make. Toolchain rovněž obsahuje make, ale také sadu kompilátorů a utilit pro vývoj systémových aplikací, konkrétně jazyky C, C++ , Fortran a Ada. Kompilátor pro jazyk Java je možné do jádra zahrnout samostatně.

Z hlediska vývoje aplikací je důležité zvolit, která sada standardů pro knihovny a hlavičkové soubory bude pro vývoj použita. Na výběr je běžně používaná „GNU C library“ (glibc) a její obdoba uClibc (nebo eglibc), která je určena pro menší zařízení s omezenou

[5] Sada prostředků pro vývojáře

správou paměti. Má omezenou funkčnost, ale zabírá méně místa v paměti, navíc je uzpůsobena pro širší spektrum architektur.

Před zkompilováním jádra je dobré vědět, jaké jádrové moduly budou pro chod požadované aplikace nutné a zahrnout je do kompilace. Aplikace a knihovny lze bez problémů dohrát do přístroje později, ale jádrové moduly musí projít křížovou kompilací, která může vést k selhání systému.

Po zkompilování jsou k dispozici obrazy systému, většinou v binárním formátu. Ty je možné nahrát přímo přes firmware výrobce nebo pomocí utilit podporující TFTP.

5.1.2 Vývojové prostředí

Po zkompilování jádra se vytvoří adresářová struktura obdobná té na cílovém zařízení. Ta obsahuje základní a zvolné knihovny, SDK a toolchain. SDK a jeho nástroj `make` je možné použít ihned, za předpokladu, že je třeba pouze upravit stávající balík aplikace s existujícím `makefile`. Pokud je třeba aplikaci vytvořit od začátku, je lepší použít některý z kompilátorů toolchain. Aby kompilátory fungovaly v systému, pro který nejsou určeny, musí se exportovat v profilu (soubor `.profile` v domovském adresáři). Pak je možné vytvářet aplikace bez `makefile` pouhým zadáním zdrojového souboru jako parametru kompilátoru v příkazové řádce. Pokud je třeba přidat knihovnu, která není součástí standardního souboru, musí se rovněž přidat jako parametr. Protože dynamické knihovny mohou mít problém s přenositelností, je lepší knihovny spojovat (linkovat) staticky, což vyžaduje zadání cesty. V (kompilací vytvořené) adresářové struktuře lze najít tyto knihovny v adresáři `staging_dir`. Samozřejmě pouze v případě, že byly vybrány pro začlenění do jádra systému.

Tímto způsobem lze tvořit jednoduché programy, které obsahují pouze několik nestandardních knihoven. V případě rozsáhlejšího projektu je vhodnější použití SDK a `makefile` ve kterém jsou uvedeny odkazy na knihovny i vlastní části projektu jako např. hlavičkové soubory.

5.2 Realizace programu

5.2.1 Výběr řešení

Vzhledem k nevhodnosti skriptovacích a interpretovacích jazyků pro náročnější aplikace zbývají pro realizaci programovací jazyky. V OpenWRT jsou k dispozici kompilátory pouze pro jazyky Java C a C++. Bohužel pro Javu není k dispozici vhodné JVM, existují sice neoficiální řešení jakým je například SableVM, ale jejich funkčnost pro současnou verzi není ověřena. Stejné problémy mají i knihovny přizpůsobené pro Javu. Zbývají tedy jazyky C a C++.

Přestože varianta socketu PF_RING měla v testu výkonnosti lepší výsledky[3] byl nakonec pro realizaci zvolen BPF společně s knihovnou pcap. Rozhodující byly především možnosti filtrování na nižších vrstvách a jejich snadné a efektivní nastavení. Protože je BPF starší řešení než PF_RING je jeho implementace podmíněna pouze knihovnou pcap. Oproti tomu PF_RING vyžaduje použití záplaty systému.

Dokumentace týkající se socketů a knihovny pcap je k dispozici především pro jazyk C. Stejně jako pro zvolenou metodu zachytávání pomocí BPF. Proto byl nakonec zvolen jazyk C.

5.2.2 Zjištění objemu dat

1) Otevření zařízení pro zachytávání [10][12]

Nejprve je v aplikaci nutné definovat síťové rozhraní, na kterém budeme naslouchat. O definici síťového rozhraní se starají mechanismy jádra systému (struktura `net_device` apd.) a samozřejmě také ovladače, v tomto případě ath9k. Jejich balíček vytvoří základní konfigurační soubor, který lze v OpenWRT nalézt jako etc/config/wireless. Zde je možné získat a měnit informace o bezdrátovém síťovém rozhraní, ovšem pouze ty, určené pro vyšší vrstvy. Vhodnější než pevně vyplněný název rozhraní je použití funkce `pcap_findalldevs()` nebo požadované rozhraní zvolit v programu.

Je také možné provádět sledování více síťových rozhraní. Při zvolení cílových rozhraní dojde k vícenásobné inicializaci knihovny. Pro otevření relace k rozhraní se použije funkce `pcap_open_live()`. Odlišení rozhraní se provede pomocí prvního vstupního parametru. Pokud chceme sledovat více rozhraní, musíme funkci volat opakovaně. Funkce vrací handler, který je pak použit jako vstupní argument pro další funkci, jenž bude zpracovávat data. Návratovou hodnotu je tedy nutné uložit i v případě, že sledujeme pouze jedno síťové rozhraní. Takto vytvořená relace už pracuje a např. pomocí funkce `pcap_loop()` je možné získávat data od rozhraní.

2) Definice filtru [2]

Jakmile se pomocí knihovny vytvoří relace pro zachytávání lze na ni aplikovat filtr. Nejprve je ho však nutné vytvořit pomocí funkce `pcap_compile()`. Po vytvoření již samozřejmě nelze měnit jeho vlastnosti, ale v programu lze vytvořit další filtry. Jeho vlastnosti se definují ve druhém parametru funkce, jako ukazatel na sled znaků ve specifickém tvaru.

Tento výraz je možné zadat v uživatelsky přívětivém formátu a to včetně IP adres. Filtr má široké možnosti nastavení, ve výrazu filtru může být definováno odchytávání veškerého provozu určitého okruhu adres nebo portů, nebo třeba jen pakety s určitou hodnotou bitu v hlavičce. Po jeho vytvoření se filtr přiřadí k rozhraní pomocí funkce `pcap_setfilter()`, kde jednomu rozhraní může být prostřednictvím handleru touto funkcí přiřazeno více filtrů.

3) Syntaxe filtrovacího výrazu [2][12]

Syntaxe závisí vždy alespoň na jednom základním filtračním kvalifikátoru. Může to být typ, směr nebo protokol. Pokud je jich použito více najednou musí být jejich vztah definován výrazem **and**, **or** nebo **not**. Zároveň může být samotný kvalifikátor složitější a obsahovat podmínky. Například pokud je třeba odchytávat pouze TCP pakety větší než 64 bajtů od a pro jednoho uživatele s IP 10.1.1.1. `tcp greater 64 and host 10.1.1.1`

4) Předání dat ke zpracování [10][12]

Protože data od rozhraní podléhají zapouzdření, nejsou k dispozici jako jeden celek. Jsou uloženy ve strukturách, které je třeba naplnit voláním funkcí knihovny pcap. Pro účely sledování provozu je vhodná funkce `pcap_loop()`, která rámce zpracovává po dávkách. Jejím parametrem je ukazatel na funkci zpětného volání označovaný jako `*(pcap_handler())`. Jedná se v podstatě o funkci která naplní definované struktury daty od rozhraní, skrze přímé přečtení z paměti, pomocí ukazatelů. Funkce zpětného volání tím určuje, kolik dat bude předáno ke zpracování na aplikační vrstvě. Bohužel to nelze provést libovolně, protože je dána minimální velikost předávaných hlaviček. U ethernetového rámce je to 14 bajtů, u segmentu i u paketu 20 bajtů.

I když jsou požadované informace načteny ve strukturách, některé je nutné upravit do uživatelsky přívětivého formátu. Například IPv4 adresy je třeba přeložit do dekadického zápisu. To se provádí pomocí funkce `inet_ntoa()`. Informace od síťového rozhraní jsou nejčastěji ve strukturách uložena ve dvou nebo čtyřbajtových celočíselných proměnných. Obecně je zde u více bajtových proměnných vždy potřeba provést přizpůsobení pořadí bajtů. To se provádí např. pomocí funkce `ntohl()` (**n**etwork **to h**ost), která uzpůsobí proměnnou definovanou hostitelem pro použití v síti.

5) Uložení a prvotní zpracování

Interpretace paketů se skládá ze dvou kroků. Prvním je přeložení samotných hodnot z hlaviček do uživatelského formátu. Přijatý paket je uložen v cache síťového adaptéru jako sled dat za sebou. Dále se pracuje s ukazatelem na začátek a délkou dat. V programu je tedy nutné mít definovanou strukturu, která přesně odpovídá složení paketu. Většinou jsou použity co nejúspornější datové typy. Chceme-li načíst některou hodnotu z hlavičky, použijeme ukazatel a délku dat k inicializaci dat v definované struktuře. Například pro definici ethernetové hlavičky

```
ethernet = (struct sniff_ethernet*)(packet);
```

je použit ukazatel na definovanou strukturu `sniff_ethernet` a délka `packet`. Pro přečtení konkrétní hodnoty z hlavičky pak jednoduše použijeme ukazatel ve vytvořené struktuře. Například `ethernet->ether_dhost[0]` pro přečtení prvního znaku z cílové adresy rámce (**d**estination **host**). Pokud chceme výsledek uložit do běžného datového typu např. z `u_char` do `char` musíme ještě ukazatel přetypovat nebo rovnou použít funkci, která data převádí do uživatelského formátu. Jak je tomu například u IP adresy pro verzi 4 `inet_ntoa(ip->ip_src)`. Pokud je nutné výsledky dále zpracovávat (např. srovnání s uloženou hodnotou) musí si odpovídat nejen datové typy ale i např. délky pole. Zde může vzniknout problém, protože některé funkce nevrací čistě hodnotu proměnné, ale mohou např. přidávat znaky konce řádku.

Druhým krokem při interpretaci je základní rozdělení, tedy kdo s kým komunikuje a jakým směrem. V případě bezdrátové přístupové sítě je to poměrně jednoduché, protože klienti zde zpravidla nekomunikují mezi sebou. Bohužel však nelze hned využívat IP adresy, protože program od začátku nedisponuje žádným seznamem uživatelských adres. Také cílem jsou libovolné servery v internetu. Protože je však lokální komunikace založena na protokolu vrstvy síťového rozhraní, lze v tomto případě využít MAC adresu. Každý rámec cílený na přístupový bod je brán, jako přenos směrem od uživatele. Zdrojová IP adresa je brána jako uživatelská a přenesená velikost jako „upload“. Pokud má rámec zdrojovou MAC adresu přístupového bodu je situace opačná. Všechny proměnné obsahující objem dat, nebo koeficienty k němu vztažené jsou dále realizovány jako dvourozměrné pole. Přičemž první hodnota je směr dat směrem od uživatele a druhá hodnota naopak k uživateli.

Aby bylo možné získaná data statisticky zpracovat, musí obsahovat informaci o pořadí, lépe však o čase přijetí. Pro účely měření provozu stačí čas v sekundách od začátku zachytávání. Musí se ale počítat s tím, že během jedné sekundy bude přijato více paketů v obou směrech. Informace o velikosti provozu jsou tedy během každé sekundy sčítány a

až poté předány k uložení, případně zpracování. To se děje ve funkci `packet_save()` do které se předává velikost rámce a vyhodnocený směr. Velikost rámce je získána na základě pole v hlavičce IP paketu, určující velikost samotné hlavičky i dat. K tomu je ještě nutné připočíst velikost ethernetové hlavičky, která je dnes většinou fixní a má 20 B.

```
sizecapture = ntohs(ip->ip_len) + SIZE_ETHERNET;
```

Takto získaná data z hlaviček se ve funkci `packet_save()` sčítají, dokud není zjištěna změna pomocné časové proměnné `livesec` oproti proměnné s aktuálním časem `aftrstrt`. Protože se obě proměnné mění po sekundách, definují součty objem dat za sekundu. Tyto hodnoty jsou ukládány do asociativního pole, aby byla snadno dostupná při zpracování do základního intervalu.

5.2.3 Ukládání dat

Pomineme - li ukládání do externí databáze, není v C/C++ mnoho možností jak ukládat větší množství dat. Nejsnazší je ukládání do souboru, které je vhodné především, pokud chceme získaná data zpracovat externím programem. Pokud je ale nutné data zpracovávat za běhu programu, který soubor plní, je třeba zabývat se problémem vícenásobného přístupu. Soubory navíc nemají implicitní mechanizmus předání pracovního ukazatele. Pokud tedy chceme např. načíst poslední řádek, je nutné projít celý soubor nebo napsat funkci která bude s ukazatelem pracovat. Obdobně složitá je také práce s běžným polem proměnných, kde je navíc většinou nutné paměť alokovat.

Jako nejlepší se jeví asociativní pole „multimap“ z jazyku C++. Hodnoty v něm jsou indexovány klíčem libovolného datového typu, takže je možné použít např. IP adresu v uživatelském formátu. Indexy mohou být duplicitní a je možné podle nich vyhledávat. Vhodnou vlastností je vyhledávání od konce, které umožňuje snadné zpracování posledních *n* prvků. Ke každému indexu lze přiřadit více hodnot v libovolném formátu. Pokud není zvoleno jinak, alokuje se pole dynamicky.

5.2.4 Základní interval

Data pro jednotlivé sekundy se po uplynutí stanoveného intervalu načítají z asociativního pole. Protože se do pole ukládá průběžně, je možné, že poslední hodnota nebude ta námi požadovaná. Je proto nutné, aby z celého souboru dat byly vybrány hodnoty pouze pro daný interval. Data jsou do pole jednak ukládána za sebou a také iterátor, přes který k nim přistupujeme, je řadí od nejmenší po největší podle klíče. Tím je v tomto případě čas, takže data jsou řazena za sebou podle toho, kdy jsou zachycena. Pro přistoupení k datům na vhodném místě je nutné naplnit iterátor první časovou hodnotou v daném intervalu. Pro to je použita funkce `lower_bound()`, která najde v pořadí první požadovanou hodnotu v poli, podle zadané proměnné. Poté se použije cyklus `for` k načítání jednotlivých hodnot. Jako počáteční hodnota je nastaven iterátor naplněný zmiňovanou funkcí a jako koncová hodnota konec intervalu pomocí funkce `end()`.

Hodnoty objemu dat v jednotlivých sekundách se v každém cyklu testují, zda hodnota není nižší než předchozí testovaná, pro zjištění nejnižší hodnoty v intervalu. Obdobně se zjišťuje nejvyšší hodnota v intervalu. Dále se v každém cyklu sčítají jednotlivé hodnoty pro download a upload. Po skončení cyklu se tyto hodnoty dělí velikostí intervalu, čímž se zjistí jeho průměrná hodnota. Výsledné hodnoty se ukládají do struktury a nakonec se struktura s časovou hodnotou uloží do asociativního pole k dalšímu zpracování.

5.2.5 Stochastický oscilátor

V programu je pro výpočet stochastického oscilátoru zavedena samostatná funkce `stochosc()`, která je volána hned po funkci, která ukládá data do základních intervalů. Samozřejmě pokud je splněna podmínka dostatečného množství základních intervalů pro zpracování. Ta je dána velikostí okna w_D indikátoru *K* `stochint`, velikostí základního intervalu **i** `chunkint` a pomocným koeficientem `secto`.

Podmínka je splněna pokud čas běhu programu `dutytime` dosáhne hodnoty počátečního času `starttime` zvětšeného o počet základních intervalů potřebných k naplnění okna indikátoru K (`stochint*chunkint`). Okno se dále posouvá podle velikosti základního intervalu (`chunkint*secto`). Do funkce `stochosc()` je předávána pouze hodnota koeficientu *k*, který je roven *t*/I (`aftrtstrtt/chunkint`). Samotná data jsou uložena v globálně definovaném asociativním poli `subcapdata`.

Ve funkci se, podle hodnoty koeficientu *k*, určí počáteční hodnota indexu v poli, která odpovídá uloženým základním intervalům. Je k tomu použita funkce `lower_bound()`, jež zajistí, že iterátor bude v poli přistupovat vždy nejdříve k prvnímu základnímu intervalu v daném okně. Tyto meze platí pro indikátor K, začínají indexem `startK`, a končí na konci pole (`subit != subcapdata.end()`). Je to možné protože funkce která plní pole základními intervaly bude znova volána až po skončení funkce oscilátoru. Pokud by měl být program více vláknový musel by být konec okna w_D dán výpočtem.

```
startK = chunkint*(sectosc-(stochint*stochintd)+1);
subit=subcapdata.lower_bound(startK);
```

Následuje cyklus, který prochází pole hodnot podle zadaných mezí v iterátoru. Zde jsou vybírány největší, nejmenší a koncová hodnota ze všech vzorků v okně. Ve vzorcích jsou již informace tohoto typu uloženy, takže se vybírá nejmenší (`minsam`) z nejmenších (`(*subitd).second.minn[0]`), největší z největších atd.

```
if(minsam>(*subitd).second.minn)minsam= (*subitd).second. minn;
```

Z těchto vzorků se po skončení cyklu vypočítá indikátor K.

```
stochk=(aceil(lastsam-minsam)/aceil(maxsam - minsam))*100;
```

Funkce `aceil` vrací nejvyšší vyšší celé číslo, čímž by se měl ošetřit stav kdy je v čitateli nebo jmenovateli nula. Výsledek se uloží do nového asociativního pole (`oscdata`) a je dále zpracováván jako klouzavý průměr do indikátoru D.

5.2.6 Bolinger Bands

Jak bylo napsáno výše, pro výpočet v určitém čase je nutné zjistit rozptyl a průměr v celém okně w_{BB}. Rozptyl se získává z aktuální hodnoty a klouzavého průměru a musí být k dispozici ještě před samotným výpočtem BB. Hodnota průměru je brána z funkce `subintmavg()`, ale poslední hodnotu rozptylu je nutné vypočítat. Je dána rozdílem poslední hodnoty základního intervalu a klouzavého průměru pro w_{BB} intervalů. Proto bylo vytvořeno nové asociativní pole `trenddata` s hodnotami ve struktuře `boll`. Při každém volání funkce `bollinger()` se tedy nejprve provede výpočet a dočasné uložení rozptylu do pole v čase `endB`.

```
boll.sqdv = pow((double)( (*subit).second.lstn - mavg), 2);
trenddata.insert(pair<int, trnd>(endB, boll));
```

Poté je možné zpracovat všechny hodnoty rozptylu najednou od začátku okna v čase `startB`. V cyklu se jednotlivé rozptyly sčítají a po jeho skončení se dělí w_{BB} a odmocní pomocí funkce `sqrt()`. Tím vznikne standardní odchylka.

Protože je ale hodnota jednotlivých rozptylů poměrně velká, může jejich součet už při několikasetkilobajtovém provozu dosáhnout hodnot větších než je hodnota běžné proměnné `int` (tedy 2^{16} resp. Pro unsigned int 2^{32}). Proto je použit typ proměnné `double`.

Po určení standardní odchylky se určí spodní a horní hranice Bollingerova pásma. Vše se uloží do asociativního pole a smaže se dočasný záznam s hodnotou rozptylu. Rovněž se provede uložení do souboru.

5.2.7 Promazávání asociativních polí

Protože je paměť v zařízení omezena, není možné delší dobu ukládat veškerá data do asociativního pole. Proto bylo nutné realizovat funkci, která bude vhodně odstraňovat již nepotřebné hodnoty. To je možné dvěma způsoby, jednak dealokací pole a pak promazáváním konkrétních hodnot pomoci funkce `erase()`. Při použití dealokace by bylo nutné platná data uložit do doby, než bude pole opět vytvořeno. Pro tento účel existuje jednoduchá funkce `swap()`, která zamění hodnoty jednoho pole za jiné (resp. zamění deskriptory polí). V tomto případě by tedy bylo potřeba další pomocné pole. Navíc před záměnou by bylo nutné v původním poli promazat nepotřebné hodnoty.

Protože je pole dynamická proměnná, podléhá automatické alokaci paměti, takže jsou kontrolována nevyužívaná místa a znova alokována. Dealokovat celé pole je proto zbytečné a je použita pouze funkce `erase()`, která umožňuje vymazání určitého rozsahu podle klíče, nebo hodnoty iterátoru.

Realizovaná funkce `mapdispatch()` je volána každou sekundu a kontroluje, zda je dostatek dat pro uvolnění. Při prvním volání je zjištěno s jakými intervaly (velikostmi oken) pracují jednotlivé funkce. Rozsah nadbytečných dat je následně určen tak, aby funkce měli vždy dostatečný počet hodnot pro svůj provoz. To se provádí pomocí iterátoru, jehož hodnota je nastavena na současný čas zmenšený o množinu potřebných intervalů `big`.

```
it = capdata.find((tmd) - (big));
capdata.erase(it);
```

Množina potřebných intervalů je určena podle největšího nastaveného intervalu (okna) u funkcí klouzavých průměrů, stochastického oscilátoru a BB.

5.2.8 Odhady

Extrapolace

Veškerá potřebná data jsou uložena uvnitř asociativního pole `trenddata` ve struktuře `oscdata` a jsou indexována podle času zpracování. Do struktury `oscdata` se během vykonávání programu zapisují vypočtené parametry. Protože se odhady podle jednotlivých hypotéz provádí jako poslední funkce, jsou v `oscdata` uloženy poslední hodnoty. Pro získání předposlední je nutné číst z pole multimap hodnotu pro předcházející čas. Protože zpracování dat probíhá pro každý základní interval, je nutné hledat čas předchozího intervalu

```
trendit = trenddata.lower_bound(timestim*chunkint-chunkint);
```

a podle něj nastavit iterátor, přes který bude následně přistupováno k předposledním hodnotám. Pak je možné jednoduše zjistit rozdíl nebo přírůstek

```
delta[1] = (oscdata.sma[1]-(*trendit).second.sma[1]);
```

a na jeho základě odhadnout následující hodnotu

```
oscdata.gs0[1] = oscdata.sma[1] + delta[1];
```

která bude uložena v současném čase, společně s ostatními vypočtenými parametry ve struktuře `oscdata` v asociativním poli `trenddata`.

Extrapolace podle stochastického oscilátoru

Vychází ze stejných proměnných jako v předchozím případě, ale k určení nárůstu nebo poklesu je nově využita hodnota indikátoru K `fastk`. Ta je sledována pomocí třech podmínek a na jejím základě je přičtena nebo odečtena absolutní hodnota rozdílu poslední `oscdata.sma` a předposlední hodnoty `(*trendit).second.sma`. Nebo v případě, že je hodnota indikátoru K v oblasti neutrálního chování

```
else if( oscdata.fastk[1]>20 || oscdata.fastk[1]<80)
```

je jako odhad použita poslední hodnota klouzavého průměru `oscdata.sma`.

5.3 Testování

5.3.1 Podmínky testování

Testovaný program běží na směrovači, který slouží jako přístupový bod v přístupové části sítě. Uživatelé se mohou připojovat přes bezdrátové i drátové rozhraní. Je to dáno tím, že program přijímá ethernetové rámce na linkové vrstvě a protokoly nižší vrstvy se nemusí zabývat. Rozdělení podle použitého standardu IEEE 802.11 nebo 802.3 by teoreticky mělo být možné, ale ovladače použitého síťového rozhraní Atheros to neumožňují. Pro zachytávání lze označit jen jedno rozhraní, ať už fyzické či virtuální. Provoz bude generován na bezdrátové rozhraní pouze z jedné stanice, ale programově bude simulován větší počet uživatelů.

Pro generování testovacího provozu byla nejprve použita aplikace Jmeter. Ta umožňuje simulovat velký počet uživatelů, požadujících služby serverů pomocí aplikačních protokolů, např. HTTP. Bohužel i při nastavení náhodných parametrů testovacího plánu je výsledný provoz spíše konstantní. Simulovaní uživatelé totiž vysílají požadavky ihned po tom, co je na jejich předchozí požadavek odpovězeno. Snížení počtu uživatelů tak vede k rychlejšímu zpracování požadavků stávajících uživatelů a naopak. Změna počtu uživatelů tak požadovaný náhodný faktor nepřináší. Proto bude provoz generován pseudonáhodně operátorem, pro každý test stejně v rámci chyby lidského faktoru.

5.3.2 Způsob testování

Provoz je generován náhodně, se stejnoměrným počtem obdobných poklesů a nárůstů, během zhruba stejné doby. I přes snahu o rovnoměrnost náhodných parametrů mohli při testování vzniknout jejím vlivem chyby. Bohužel, nelze přesně určit, jaká odchylka od testovacího plánu se podílela na chybě v hypotéze. Proto, nebudou tyto chyby vyčíslovány a jejich vliv bude pouze shrnut v závěru.

U každé hypotézy bude krom chybovosti sledována také platnost hypotézy. Oblast přípustných hodnot je vymezena Bollingerovým pásmem, které vychází ze standardní odchylky. Protože standardní odchylka se počítá z průměru o *n* prvcích, musela by být chyba odhadu v jednom základním intervalu *n* násobně větší než průměrná standardní odchylka v daném okně. Vzhledem k tomu, že všechny budoucí hypotézy vychází z hodnot průměrů, je pravděpodobnost shledání hypotézy jako neplatné, malá.

Velikost základního intervalu, stejně jako délka okna pro každou funkci se dá nastavit. Pro korektnost výsledků by měly všechny související funkce vycházet ze stejného souboru

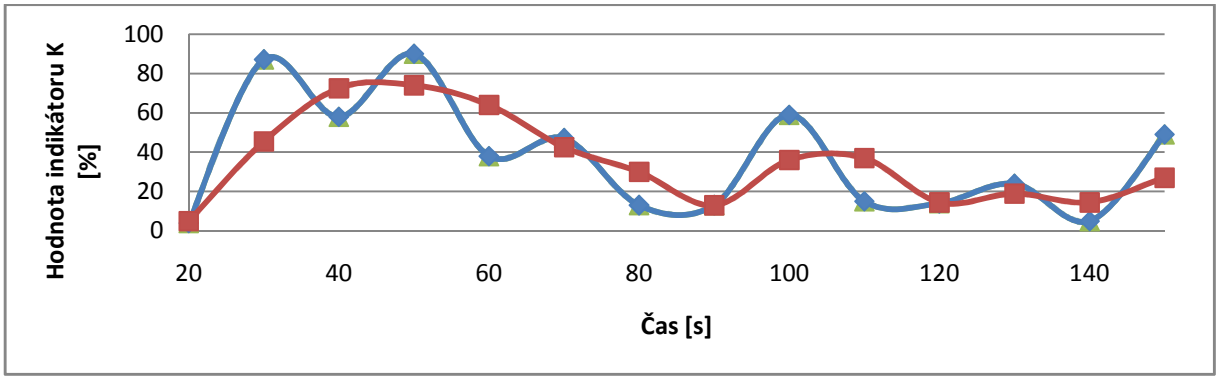
dat. Základní interval je pro všechny funkce vždy stejný, ale dále je nutné, aby měli nastavenou stejnou velikost okna, která se nastavuje pro každou funkci před spuštěním testu. Testování tedy bude prováděno pro rozdílné velikosti, základního intervalu a okna. V každém testu budou průběžně určovány chyby jednotlivých odhadů a nakonec celková chyba. Použita bude relativní chyba, která univerzálněji vyjadřuje odchylky pro rozdílně veliké hodnoty. Na závěr kapitoly budou veškeré testy porovnány a zhodnoceny.

5.3.3 Stochastický oscilátor

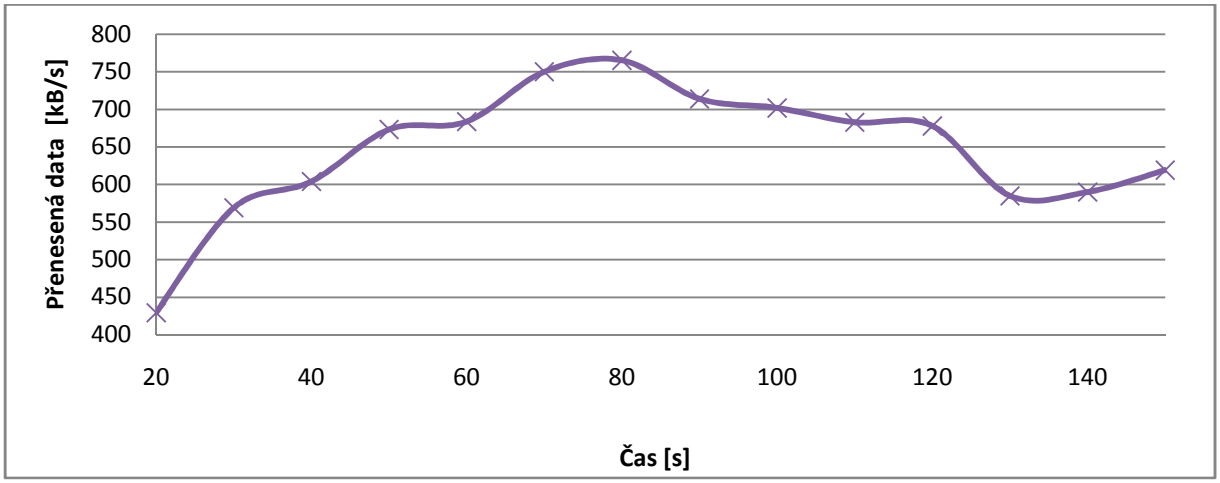
Jak již bylo napsáno v teoretické části a v literatuře [15], indikátory této metody nemají vždy stoprocentní platnost. Proto bylo provedeno několik testů k ověření úspěšnosti.

Stochastický oscilátor se vztahuje k datům základnímu intervalu. Rozhodující jsou zde největší a koncový objem dat přenesený za sekundu, v daném časovém intervalu. Jeho výsledky ale budou využívány v souvislosti s klouzavým průměrem použitým k odhadu. Tato skutečnost sama o sobě zvyšuje pravděpodobnost možné chyby. Dále se šance na chybný odhad se zvětšuje s velikostí okna. Stačí totiž jeden extrémní výkyv maximální hodnoty a u veškerých oken, které ho obsahují, dojde k znehodnocení výsledku.

Následující grafy ilustrují jeden z možných průběhů s lepšími výsledky. Použity jsou hodnoty objemu dat směrem k uživateli. Ty byly během všech testů mnohem větší, než naopak a mají proto větší vypovídací hodnotu. Základní interval má délku 10 sekund a velikost okna stochastického oscilátoru i klouzavého průměru je 2. První hodnota klouzavého průměru je tak k dispozici v čase 20 sekund. Na Obr. 5.1 je modře Indikátor K a červeně indikátor D, ten je nastaven na hodnotu 2. Jde tedy o klouzavý průměr hodnot indikátoru K s velikostí okna 2.



Obr. 5.1: Indikátor K a D



Obr. 5.2: Jednoduchý klouzavý průměr

Jak je vidět na obr. 5.2 zpočátku průměr narůstá až do času 80s. V tomto úseku lze napočítat 6 narůstajících hodnot. Indikovány jsou však pouze hodnoty v čase 30s a 60s pomocí indikátoru K který má velikost větší než 80%. S klesajícími hodnotami je situace o něco lepší, z pěti hodnot jsou indikovány čtyři. V uvažovaném případě tedy byla úspěšnost odhadu vývoje trendu zhruba poloviční. Navíc dále klesá, pokud se podíl narůstajících hodnot v rozsahu testu zvětšuje.

I přes neutrální chování v případě hodnoty indikátoru s malou vypovídací hodnotou (20%-80%) ani tato metoda není bezchybná. To je vidět ve stočtyřicáté sekundě, kdy má být indikován pokles a nikoliv nárůst. Chyba se dále zvětšuje v případě konstantního provozu, kde je velká pravděpodobnost, že koncová hodnota bude blízká nebo shodná s maximální.

Realizován byl rovněž indikátor D, který je v podstatě průměrem několika indikátorů K. V případě síťového provozu ale neplní očekávanou funkci, indikace významných odchylek, a spíše ještě zvyšuje již tak dost vysokou chybu indikátoru K. Pokud by byly upraveny meze pro indikátor D, byl by použitelný pro rovnoměrně stoupající nebo klesající trendy, nebo i pro konstantní provoz, kde by filtroval nežádoucí odchylky indikátoru K. Takovéto ideální podmínky, však nelze očekávat.

Úspěšnost této metody bude ještě dále otestována použitím indikátoru K při odhadech, ale je zjevné, že bude přínosná pouze v určitém úzkém spektru podmínek. Proto je vhodné ji zkombinovat s jinou metodou, která bude její nedostatky minimalizovat.

5.3.4 Extrapolace

Protože jsou data rozdělena do pevně daných intervalů v čase, lze na ně nahlížet jako na dvourozměrný statistický soubor. Kde jedním rozměrem je objem dat a druhým čas, přičemž se předpokládá, že objem dat je závislý na čase a na předchozí hodnotě objemu. Protože je ale čas daný, lze tento stav zjednodušit přidáním výchozí podmínky, která předem určí vývoj časových argumentů. Hodnota odhadovaného objemu dat tedy bude záviset pouze na předchozích hodnotách objemu. Z toho je patrné, že po zjednodušení půjde pouze o jednorozměrný statistický soubor.

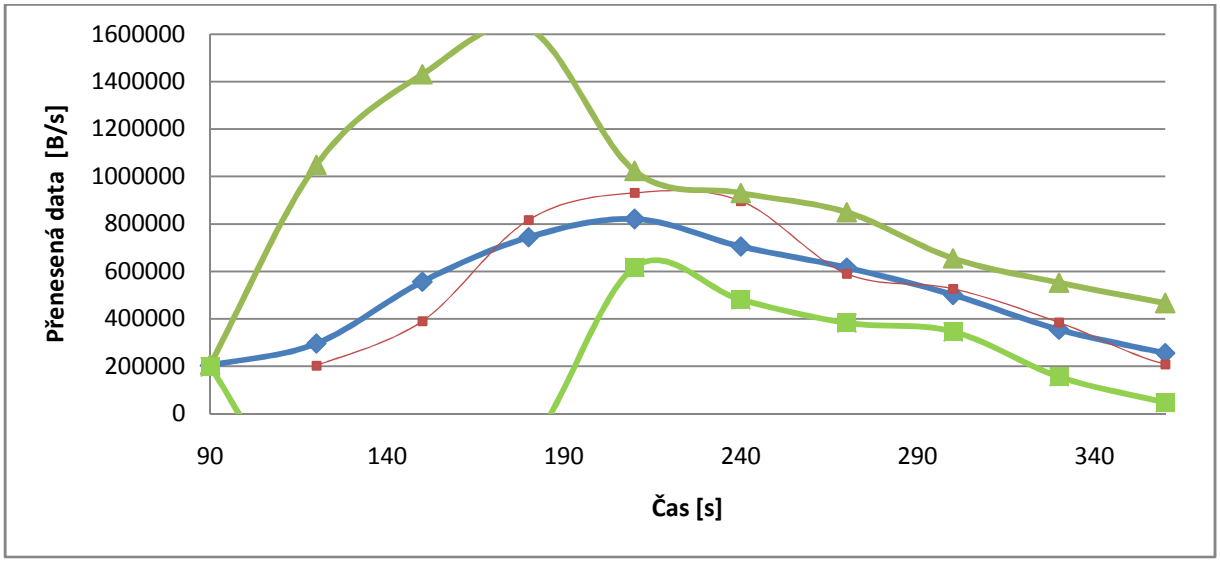
Pro odhad neznámé budoucí hodnoty objemu provozu bude zřejmě nejvhodnější lokální extrapolace. Objem se totiž dynamicky mění a nelze očekávat, že použití většího množství hodnot povede ke zpřesnění. Jako základní (alternativní) hypotéza tak byla zvolena extrapolace. V kombinaci s pouze jednorozměrným souborem lze zjednodušit vzorec 2.10b pro lineární aproximační funkci. Protože na ose x je konstantně narůstající čas, bude rozdíl po sobě jdoucích hodnot vždy stejný. Což znamená stejné číslo v čitateli i jmenovateli zlomku, a jeho výsledkem tak bude vždy jednička. Po úpravě tedy

$$y = y_N + (y_N - y_{N-1}) \tag{5.1}$$

kde y představuje odhadovanou hodnotu, y_N poslední známou a y_{N-1} předposlední známou.

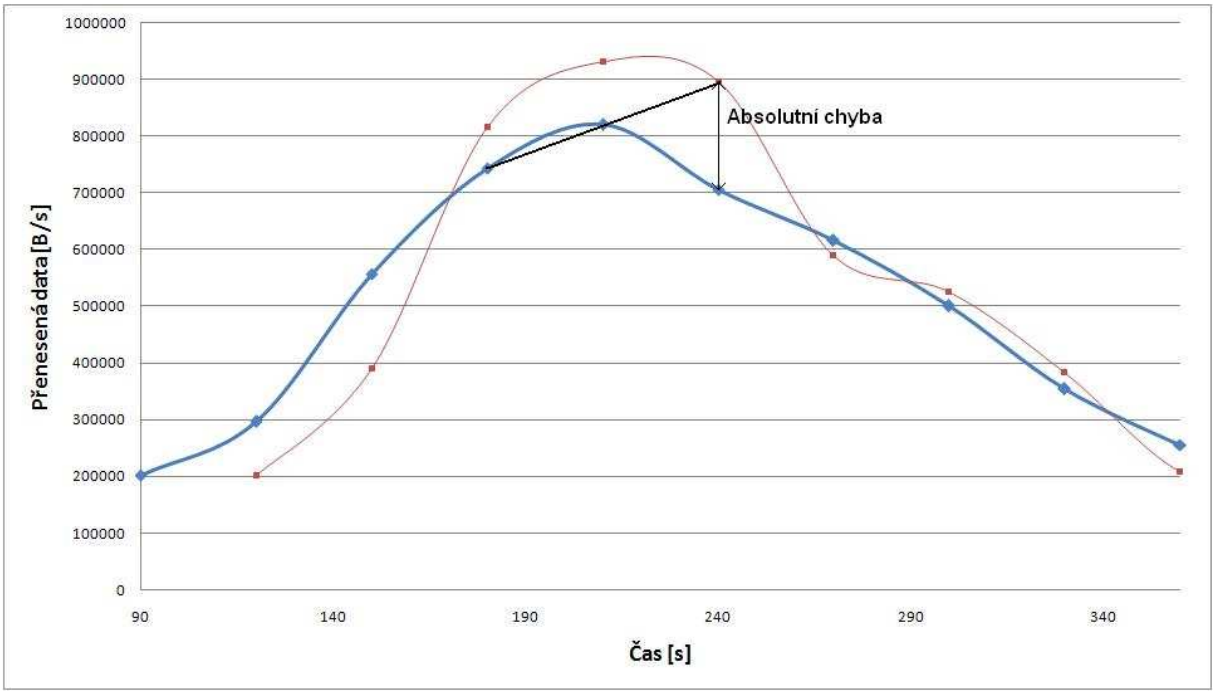
Pro samotný výpočet bude použit hodnot klouzavého průměru v intervalu s nastavitelnou délkou. Ten se posouvá tak aby jeho poslední hodnota byla vždy poslední zpracovaná. Nevychází se zde ze všech vzorků, ale z několika předzpracovaných hodnot základního intervalu. V tomto případě se používá průměrná hodnota objemu dat za sekundu. To, že jsou zachycená data zprůměrována dvakrát za sebou, samo o sobě zmenšuje prudké odchylky, které jsou pro extrapolaci nežádoucí.

Pro ověření platnosti hypotézy byl zvolen test s největší průměrnou chybou pro okno 3 a délku základního intervalu 30 sekund. Na obr. 5.3 je červeně odhadovaný průběh, modře klouzavý průměr a zeleně hranice bollingerova pásma, vymezující oblast přípustných hodnot. Je vidět, že hodnoty odhadu toto pásmo neopustili a hypotéza je proto platná.



Obr. 5.3: Extrapolace v Bollingerově pásmu

Chování extrapolace splnilo očekávání. K chybám dochází při změně tempa růstu nebo poklesu. Největší rozdíl od průměru bývá v dalším intervalu po obratu trendu. Pro stejné hodnoty jako v předchozím případě byl vytvořen graf na obr. 5.4. Zde je vyznačena přímka procházející dvěma body před obratem trendu a z nich vycházející odhad. Vyznačená absolutní chyba jde od 700 kB/s do 900 kB/s, a relativní chyba je tedy zhruba 30%. V tomto konkrétním případě by bylo výhodnější použít druhý způsob lineárního extrapolace. Ten jednoduše počítá s tím, že neznámá hodnota bude stejná (nebo blízká) té poslední.



Obr. 5.4: Příklad absolutní chyby extrapolace

V grafu je dále vidět, že k větším chybám dochází v první polovině, kdy graf narůstá. Chyby nejsou dány samotným nárůstem, ale tím, že v této části je vývoj méně lineární.

Dále provedené testy měli za úkol zjistit jaký vliv má nastavení délky základního intervalu a velikosti okna, klouzavého průměru a stochastického oscilátoru, na průměrnou chybu. Použity byly tři různé velikosti základního intervalu 3, 15 a 30 sekund. Původně měli být i tři velikosti okna, ale ukázalo se, že testovací plány s obdobnými vlastnostmi lze jen obtížně replikovat pro hodně rozdílné doby trvání. Proto byly zvoleny hodnoty 5, 15 a 25, ale pro třicetivteřinový základní interval byly použity hodnoty 3, 4 a 5. Tím je sice znemožněno přímé srovnání, ale stále je možné určit úměrnost délky použitého intervalu na průměrnou chybu.

Tab. 5.1 Průměrná relativní chyba extrapolace s různými vstupními parametry

Download [%]

	Základní interval [s]		
Velikost okna [-]	3	12	30
5 (3)	6.6470	13.4531	27.8331
15 (4)	3.7279	4.6426	22.2543
25 (5)	4.4446	3.4018	20.5424

Upload [%]

Základní interval [s]		
3	12	30
4.2414	5.9224	14.8982
2.2662	3.6536	12.7205
2.6625	2.5035	11.7420

Z tab. 5.1 je vidět, že se zvyšující se velikostí okna se průměrná chyba snižuje. Naopak větší základní interval znamená větší průměrnou chybu. Ve dvou případech toto tvrzení neplatí. Například u třísekundového intervalu s oknem 15. Tuto pozitivní odchylku lze připsat průběhu testu s příliš lineárními nárůsty a poklesy. Takovéto nastavení mělo lepší úspěšnost pouze v konkrétním testovacím průběhu. Nelze vyvozovat závěry, že např. tento poměr vstupních parametrů je všeobecně lepší než ostatní. Obecně lze říci, že větší délka okna znamená více informací pro určení jedné odhadované hodnoty.

5.3.5 Kombinovaná metoda 1

U samotné extrapolace dochází k největším chybám v bodech, kde se obrací vývoj trendu. Protože byla realizována funkce stochastického oscilátoru, může být k odhadnutí těchto bodů použit indikátor K. Jeho hodnotu lze vyhodnocovat pro tři možné scénáře.

1) Hodnota menší než 20% signalizuje budoucí nárůst.

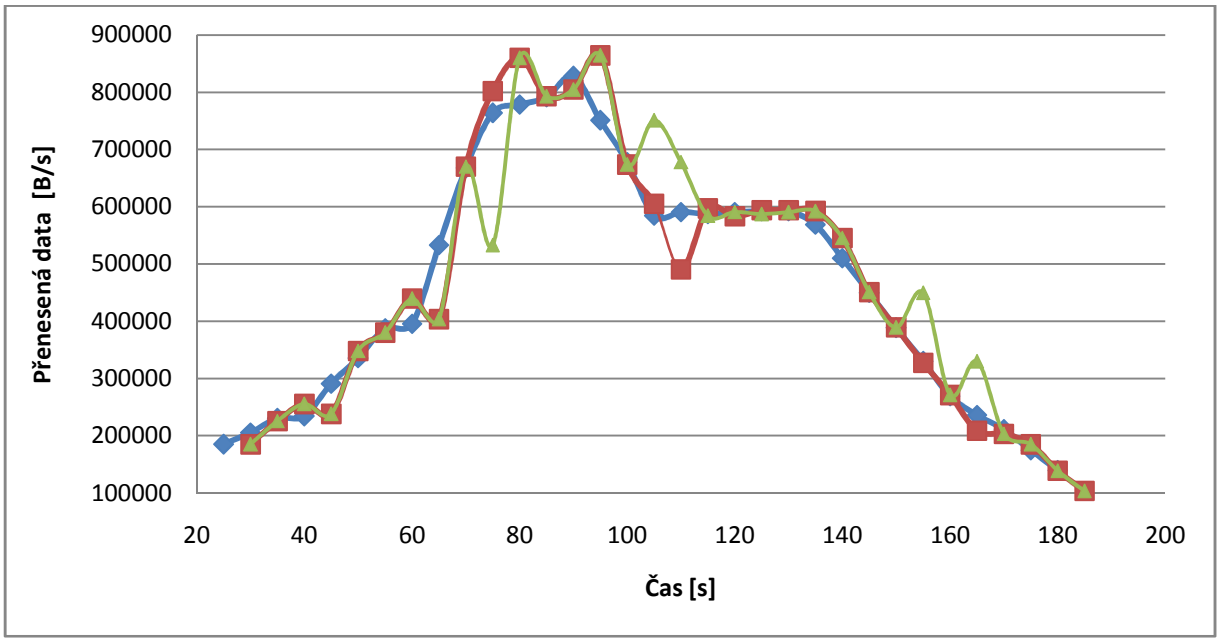
2) Hodnota mezi 20% a 80% má malou vypovídací hodnotu a nebude brána v úvahu.

3) Hodnota větší než 80% indikuje budoucí pokles.

V případě 2) může dojít k výraznému poklesu i nárůstu, který není detekován. Vzhledem k úspěšnosti stochastického oscilátoru má tato možnost za ideálních podmínek zhruba poloviční pravděpodobnost. Proto bude v tomto případě zatím použita běžná extrapolace.

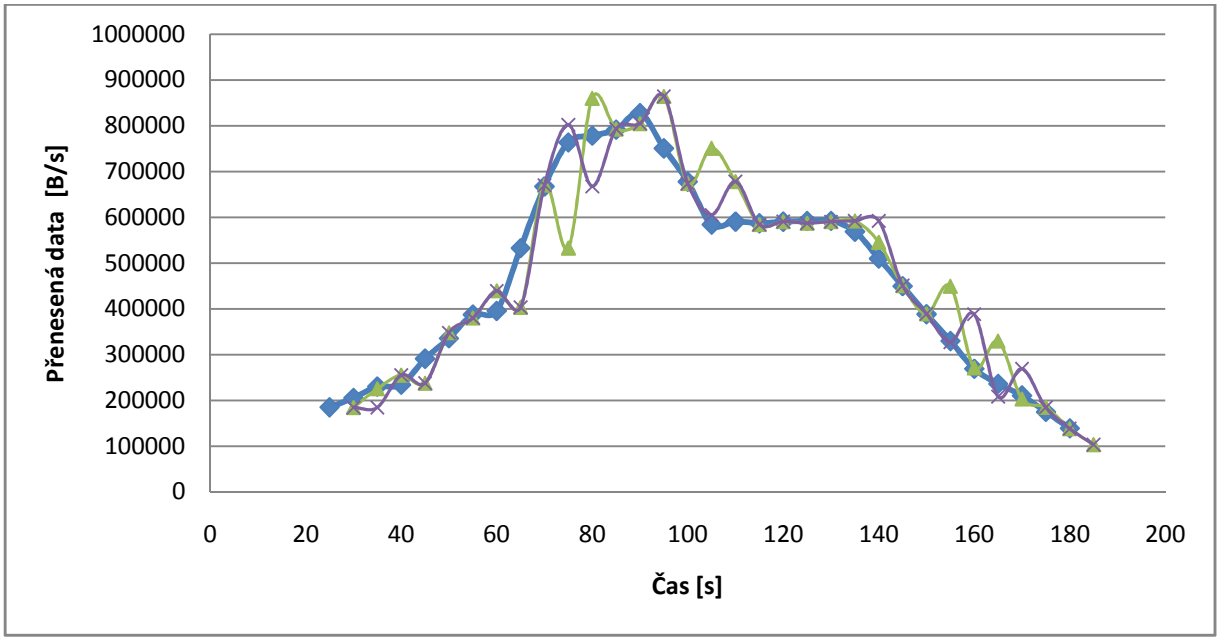
Nárůst a pokles odhadovaných hodnot bude prováděn obdobně jako u extrapolace. Takže v případě 1) bude přičtena absolutní hodnota rozdílu poslední a předposlední hodnoty klouzavého průměru. Obdobně budou tyto hodnoty odečteny v případě 3).

Následující graf na obr. 5.6 znázorňuje realizovanou metodu zeleně, extrapolaci červeně a klouzavý průměr modře. Uvedený průběh měl pětisekundový interval a rovněž okno mělo hodnotu pět. V uvedeném průběhu nedošlo aplikací stochastického oscilátoru ke zlepšení ani v jednom bodě. Podíl na tom má chyba stochastického oscilátoru, která se projevila v časech 75 s, 105 s 110 s, 155 s a 165 s. Navíc místo indikace nárůstu došlo ve stodesáté sekundě k indikaci poklesu a ve zbývajících případech došlo k jevu opačnému. K očekávanému odfiltrování chyb v bodech obratu trendu tedy nedošlo, a naopak se zvýšila průměrná chyba. Jako řešení se jeví otočení intervalů pro indikaci stochastického oscilátoru, tedy do 20% indikace poklesu a od 80% indikace nárůstu. Pokud by ale v tomto případě nedošlo k chybám, byl by výsledek naprosto stejný jako samotná extrapolace.



Obr. 5.6: Zohlednění stochastického oscilátoru při extrapolaci

Při tvorbě programu byla omylem realizována metoda, která nepočítala se současnou hodnotou indikátoru K, ale s hodnotou zpožděnou o jeden základní interval. Překvapivě vedla k lepším výsledkům než metoda původní. Navíc v místech obratu trendu občas vykazovala opačnou chybu než současná metoda, jak je vidět na obr. 5.7 v čase 85 s.



Obr. 5.7: Metody s poslední a předposlední hodnotou indikátoru K

Aby bylo možné dosáhnout lepších výsledků, bylo nutné minimalizovat chybu danou špatným odhadem a rovněž zmenšit chybu v bodech obratu.

Metoda kombinace extrapolace se stochastickým oscilátorem tedy byla upravena. Nově se pro určení nárůstu nebo poklesu využívá dvou hodnot indikátoru K, předposlední a poslední. Nejprve se určí odhadované hodnoty pro oba indikátory a poté se zprůměrují. V případě indikace poklesu nebo nárůstu je přičtena absolutní hodnota rozdílu předchozích dvou bodů.

Nově realizovaná metoda má obdobnou průměrnou chybu jako extrapolace ve všech testech. V posledním realizovaném testu (30 s, okno 5) je zhruba o půl procenta lepší než samotná extrapolace. Kladný nebo nulový rozdíl je také ve všech testech pro délku základního intervalu 12 s.

Tab. 5.2 Průměrná relativní chyba kombinované metody

Download [%]

	Základní interval [s]		
Velikost okna [-]	3	12	30
5	7.6116	13.3784	28.8954
15	3.7217	4.6426	21.7471
25	4.4670	3.3286	20.0742

Upload [%]

Základní interval [s]		
3	12	30
4.2412	6.9063	15.8904
2.2659	3.6536	12.0644
2.7047	2.5102	11.1363

5.3.5 Kombinovaná metoda 2

Protože indikátor K vychází z rozdílu velikosti vzorků v základním intervalu, lze očekávat, že jeho chyba bude růst s extrémně malými nebo velkými[6] odchylkami od průměru. Realizovaná standardní odchylka umožňuje výpočet tzv. variačního koeficientu *v*, který míru odchylek vyjadřuje procentuálně. Přičemž čím vyšší procento, tím větší míra odchylek se pravděpodobně podílela na vzniku standardní odchylky. Bohužel však nelze rozlišit, jestli odchylku tvoří jeden extrémně velký nebo více velkých výkyvů.

Chybu první kombinované metody by mělo být možné snížit zahrnutím variačního koeficientu ve výpočtu. Konkrétně by měl zmenšovat chybu v případě nesprávného odhadu vlivem extrémních maximálních výkyvů. Standardní odchylka vypočítávaná pro bollingerovo pásmo se určuje ze stejného časového okna jako stochastický oscilátor. Proto se na odchylce projeví i jeden extrémní výkyv v okně, který může způsobovat chybu indikátoru K.

Protože odhad vychází z absolutní hodnoty rozdílu poslední a předposlední hodnoty klouzavého průměru, bude tento rozdíl zmenšen o procentuální hodnotu variačního koeficientu. Zjednodušeně tak lze popsat odhad pro nárůst

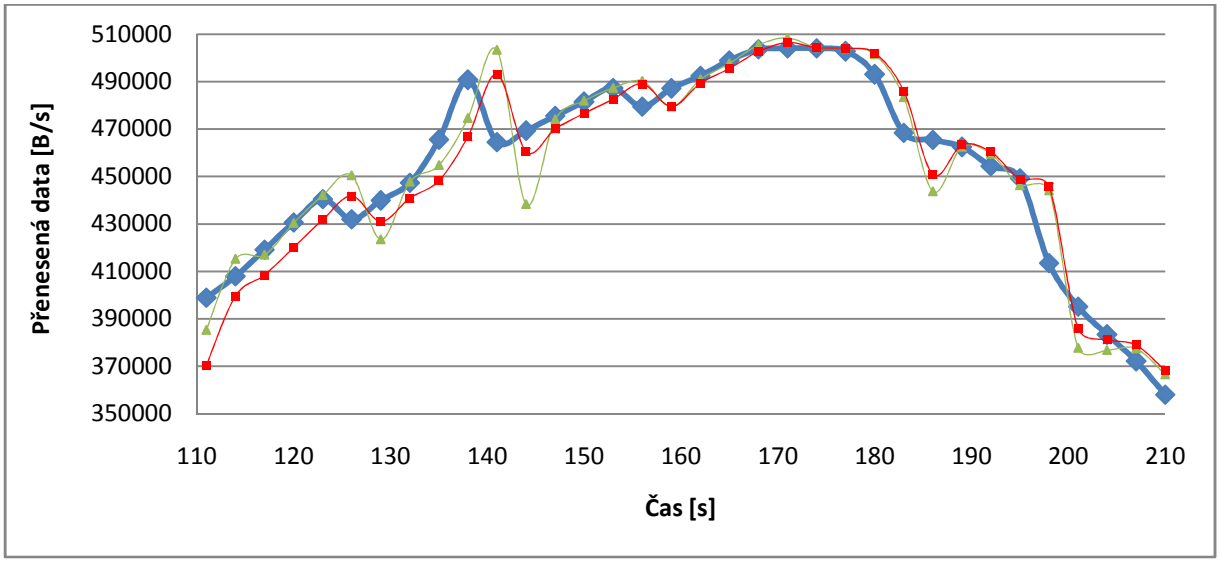
$$y = y_N \pm \left| y_{N-1} - y_N \right| \cdot (1 \pm v_N) \tag{5.2}$$

kde *y* je odhadovaná hodnota a v_N je variační koeficient v posledním základním intervalu a hodnoty jsou odčítány nebo přičítány na základě indikátoru K.

Výsledky testů této metody potvrdili platnost metody jako hypotézy, protože nabývala hodnot z přípustného pásma hodnot daného standardní odchylkou. Očekávání ale bylo splněno pouze částečně.

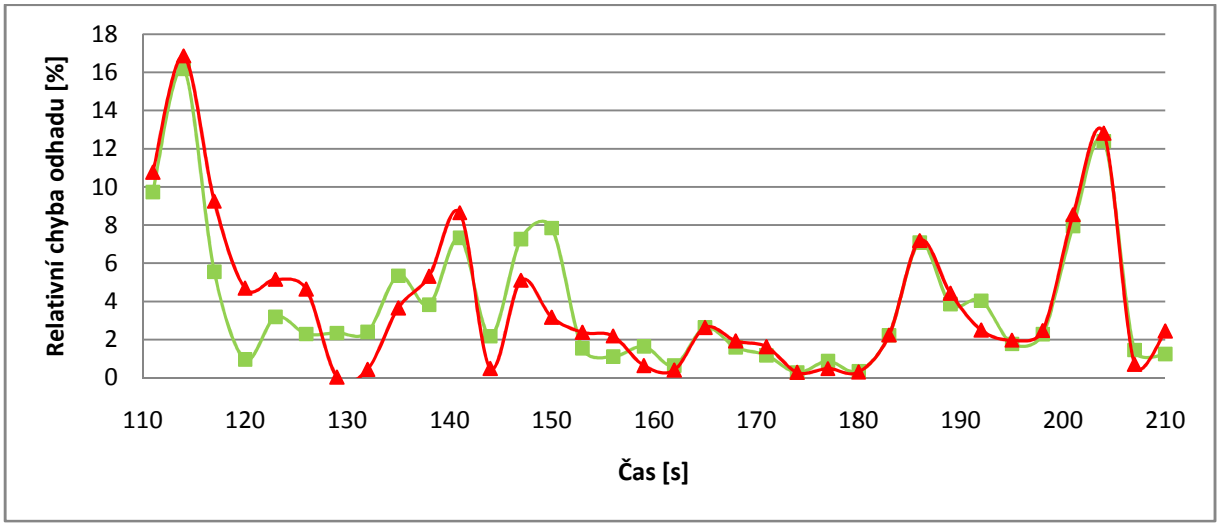
Bylo dosaženo zmenšení chyby v bodech obratu vývoje trendu, jak je vidět na obr. 5.8. Kde je modře klouzavý průměr, zeleně první kombinovaná metoda a červeně nová metoda. Dále zde lze pozorovat chybu, kterou tato nová metoda přinesla. Ta je nejmarkantnější v základních intervalech, ve kterých nedochází k bodu obratu, ale kde má klouzavý průměr velkou standardní odchylku.

[6] V testech se běžně objevovali výchylky s velikostí zhruba dvojnásobnou, oproti průměru



Obr. 5.8: Srovnání první a druhé kombinované metody

Teoreticky by nová metoda měla vykazovat větší počet menších chyb odhadu. Proto byl průběh chyb vynesen do grafu na obr. 5.9 pro snadnější porovnání. Zde je jasně vidět, že tento předpoklad nebyl splněn. Podíl menších a větších chyb je zhruba stejný, ale většinou je velikost téměř totožná. Vzhledem k tomu, že byl vybrán test s nejmenší chybovostí, nelze ani v ostatních případech očekávat lepší vlastnosti.



Obr. 5.9: Srovnání chyb první a druhé kombinované metody

Protože velká standardní odchylka automaticky neznamená bod obratu v grafu, byl počet odhadů chybně snížených variačním koeficientem větší, než množství odhadů snížených správně. Navíc relativně rovnoměrný průběh měl v testech větší podíl než body obratu. Výsledná chyba je tedy ve všech testech větší než původní kombinovaná metoda.

I když je druhá kombinovaná metoda platná, neměla v žádném prováděném testu chybovost menší než předchozí realizované metody, jak je vidět v Tab 5.3. V některých

případech filtruje odchylky v bodech obratu, ale tato skutečnost je vyvážena větší průměrnou chybou. Použitelnost této metody je tedy velice nízká.

Tab. 5.3 Průměrná relativní chyba druhé kombinované metody.

Download [%]

	Základní interval [s]		
Velikost okna [-]	3	12	30
5	7.6987	17.4539	31.5629
15	3.9722	4.5996	22.9700
25	5.3890	3.7929	21.2031

Upload [%]

Základní interval [s]		
3	12	30
4.1999	7.1059	14.9576
2.2379	3.1227	12.5819
2.6884	2.4087	11.6140

Tato metoda odhadu má pozitivní výsledky pouze v bodech obratu vývoje trendu, v případě velké standardní odchylky. Tento faktor může být neutralizován současně s velikostí samotných odchylek. Pokud bude například omezeno přidělené přenosové pásmo, bude s narůstajícím objemem provozu klesat takto shora omezená odchylka.

7 ZÁVĚR

V první části práce byly zhodnoceny hardwarové parametry dostupných přístupových bodů a jejich možný vliv na výkonnost realizované aplikace. Dále byl vybrán firmware, který poskytnul vhodné prostředí pro běh a vývoji samotné aplikace. Na výběru OpenWRT pak závisely další volby nutné pro vývoj programu. Konkrétně dostupné jazyky byly omezeny podporovanými vývojovými prostředky, proto byl zvolen jazyk C/C++. Na základě vlastností, zkušeností z obdobných programů a dostupné dokumentace byla pro realizaci zachytávání provozu zvolena knihovna pcap. Poté byl vytvořen jednoduchý program, jehož úspěšným otestováním byla potvrzena vhodnost zvoleného řešení.

Druhá část práce spočívala ve zpracování získaných dat ke dvěma cílům. Prvním bylo vytvoření statistiky a druhým realizace metody pro odhad vývoje objemu provozu. Proto byly realizovány mechanizmy z technické analýzy a následně použity v hypotézách pro odhady.

Pro ověření hodnot velikostí rámců, načítaných prostřednictvím knihovny pcap, byl použit program wireshark. I když se během realizace zdálo, že dochází ke stavům, kdy jsou získávána nekorektní data, testy to nepotvrdili.

Všechny zadané metody pro zpracování se podařilo úspěšně realizovat, pouze exponenciální klouzavý průměr poskytoval mírně zkreslené výsledky. Trvalo vždy několik základních intervalů, než začaly být produkovány věrohodné výsledky. Celý průběh exponenciálního klouzavého průměru pak byl v závislosti na nastavené délce okna o několik procent větší nebo menší než hodnota jednoduchého klouzavého průměru. Kvůli těmto nedostatkům byl pro odhady použit jednoduchý klouzavý průměr.

Jako základní (alternativní) hypotéza pro odhady byla zvolena ověřená metoda lineární extrapolace. Platnost hypotéz byla ověřována prostřednictvím oblasti platných hodnot, realizovanou bollingerovým pásmem. Veškeré realizované hypotézy produkovaly ve všech provedených testech hodnoty v rámci této oblasti. Všechny tedy byly přijaty jako platné.

V práci byly prezentovány dvě navrhnuté hypotézy pro odhad následující hodnoty objemu provozu. Obě dvě vycházeli z hodnot indikátoru K, získaných mechanizmem stochastického oscilátoru. Ten sám o sobě přináší podstatnou chybu, ale v kombinaci s extrapolací produkuje srovnatelné hodnoty. Za určitých podmínek dokonce i o málo lepší.

Nárůsty i poklesy klouzavého průměru v simulovaném provozu jsou z větší části dány prudkými odchylkami nebo skokovou změnou generovaného testovacího provozu. To je

problém pro stochastický oscilátor, který indikuje pokles nebo nárůst podle jednotlivých vzorků objemu dat. Ve snaze vykompenzovat tuto skutečnost byla navržena další hypotéza.

Ta byla založena na faktu, že chyby stochastického oscilátoru jsou z části způsobeny velkými výkyvy oproti průměru. Protože se ale takovýto výkyv projevuje pro standardní odchylky v celém okně stochastického oscilátoru, byla výsledná průměrná relativní chyba větší než u předchozí kombinované metody. Tedy i nová metoda, měla ve všech testech větší průměrnou relativní chybu, než samotná lineární extrapolace.

Navržené způsoby odhadu budoucího objemu provozu lze označit jako použitelné, ale ve většině případů vykazuje lineární extrapolace lepší výsledky. To může být dáno výchozími podmínkami, protože testovací průběhy provozu byly generovány poměrně rovnoměrně. Také použitý jednoduchý klouzavý průměr má tendenci zvyšovat linearitu náhodných vzorků, což je pro extrapolaci přínosem.

Metody zpracování i odhadů byly realizovány přímo na přístupovém bodě. V případě potřeby by tak bylo možné výsledky použít k předcházení zahlcení přístroje, nebo k rozložení zátěže iniciací handoveru uživatele k méně vytíženému přístupovému bodu. Možnosti vizualizace výsledků jsou ale na přístupovém bodu omezené. Vykreslovat grafy by bylo možné ve webovém rozhraní. Pak by bylo potřeba vytvořit modul pro rozhraní luci, které používá OpenWRT. Pro použití grafických prostředí navrhovaných v zadání by bylo nutné získaná data pomocí vhodného rozhraní nejprve přenést na PC. Běžně by se dal použít například skript s využitím programu scp, ten se ale ve zjednodušeném shellu -ash nepodařilo realizovat. Data proto byla ručně importována do programu MS Excel. Ten umožňuje variabilnější zpracování, než případné knihovny či moduly pro navrhovaná vývojová prostředí.

Kromě grafické interpretace získaných dat byly splněny veškeré podmínky zadání v celém rozsahu. Vybraná platforma OpenWRT na zvoleném směrovači s architekturou Atheros umožnila vývoj aplikace pro zachytávání provozu. Realizovaný program byl stabilní během všech provedených testů a pravdivost produkovaných dat byla ověřena programem wireshark. Dále se podařilo úspěšně realizovat zadané metody klouzavých průměrů, Stochastického oscilátoru a Bollingerova pásma. Jako nejúspěšnější metoda odhadu vývoje provozu se i přes snahy o vylepšení ukázala prostá lineární extrapolace.

SEZNAM ZDROJŮ

[1] NEMETH, E., SNYDER, G., HEIN T. Linux - Kompletní příručka administrátora. Computer Press, 2004. 880 s. ISBN: 80-722-6919-4.

[2] STEVENS, Richard. UNIX Network Programming Vol1 : Networking API´s: Socket and XTI. [s.l.] : [s.n.], 2003. 1024 s. ISBN 0-13-141155-1.

[3] BRAUN, Lothar. ComparingandImprovingCurrent Packet Capturing : Solutionsbasedon Commodity Hardware [online]. [s.l.], 2010. 12 s. Oborová práce. Technische Universität München. Dostupné z WWW: <conferences.sigcomm.org/imc/2010/papers/p206.pdf>. ACM978-1-4503-0057-5/10/11.

[4] DERI, Luca. *Exploiting Commodity Multi-core Systems for Network Traffic Analysis* [online]. 2010. 11 s. Oborová práce. University of Pisa . Dostupné z WWW: <http://luca.ntop.org/imc2010.pdf>.

[5] DONAHO, Michael. *TCP/IP Sockets in C*. Second Edition. [s.l.] : [s.n.], 2009. 216 s. ISBN 0123745403.

[6] DERI, Luca. *Improving Passive Packet Capture : Beyond Device Polling* [online]. 2004. 12 s. Oborová práce. University of Pisa. Dostupné z WWW: <http://luca.ntop.org/Ring.pdf>.

[7] CORBET, Jonathan; RUBINI, Alessandro. *Linux Device Drivers* [online]. Third Edition. [s.l.] : O´Reilly, 2005 [cit. 2010-12-13]. Dostupné z WWW: <http://lwn.net/images/pdf/LDD3/>. ISBN 0-596-00590-3

[8] *The Linux Foundation : Networking* [online]. 2009 [cit. 2010-12-13]. www.linuxfoundation.org. Dostupné z WWW: <http://www.linuxfoundation.org/collaborate/workgroups/networking/napi>.

[9] MCCANNE, Steven. *The BSD Packet Filter* [online]. [s.l.], 1992. 11 s. Oborová práce. Lawrence Berkeley National Laboratory. DE-AC03-76SF00098.

[10] JACOBSON, Van ; LERES, Craig ; MCCANNE, Steven. *Linux Documentation* [online]. 2004 [cit. 2010-12-14]. Packet Capture Library. Dostupné z WWW: <http://linux.die.net/man/3/pcap>.

[11] BAKER, Mike, et al. <http://openwrt.org> *: Wireless Freedom* [online]. 2010 [cit. 2010-12-14]. Dostupné z WWW: <openwrt.org>.

[12] CARSTENS, Tim ; HARRIS, Guy . *TPCDUMP/LIBPCAP* [online]. 2002 [cit. 2010-12-14]. Dostupné z WWW: <http://www.tcpdump.org/pcap.html>.

[13] ATHEROS COMMUNICATIONS AR7100 Wireless NPU Technology Backgrounder : Network Processor Strategy. In [online]. [s.l.] : [s.n.], 2007 [cit. 2010-12-14]. Dostupné z WWW: <http://www.atheros.com/media/resource/resource_23_file2.pdf>. Doc-Number 991-00010-001.

[14] KUBANOVÁ, Jana. *Matematická Statistika*. 1. Pardubice : Universita Pardubice, 1999. 107 s. ISBN 80-7194-215-4.

[15] ELDER, Alexander. *Tarding for living*. 1 edition. [s.l.] : [s.n.], 1993. 289 s. ISBN 978-0471592242.

[16] HOMOLA, Vladimír . Interpolace a extrapolace v rovině. 1. Ostrava. *VŠB,* 2002, Dostupný z WWW: <http://homel.vsb.cz/~hom50/SLBSTATS/IER/GS03.HTM>.

SEZNAM ZKRATEK

API	Application Programming Interface
BPF	BSD Packet Filter
BSD	Berkeley Software Distribution
DMA	Direct Memory Access
HAL	Hardware Abstraction Layer
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
JRE	Java Runtime Enviroment
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LSB	Least significant byte
MIPS	Microprocessor without Interlocked Pipeline Stages
NAPI	New API
NIC	Network Interface Controller
NPU	Network Processing Unit
PS	Protocol Stack
RISC	Reduced Instruction Set Computer
SW	Software
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TNAPI	Threaded NAPI
TTL	Transistor Transistor Logic

PŘÍLOHA

Kompilace OpenWRT

Veškeré uvedené cesty začínají `home/uzivatel/`

Veškeré operace se provádějí pod uživatelským účtem, nikoli jako root. V případě nutných rootovských práv použijte příkaz `sudo`. Pokud se vám při instalaci vytvořil uživatel, který není uvedený v sudoers spusťte terminál uživatele root (v debianu) a příkazem `visudo` otevřete sudoers. Zde dopište řádek `uzivatel ALL=(ALL) ALL` za obdobný řádek se superuživatelem root (mezi uzivatel a ALL je nutné použít tabulátor).

Nejprve je nutné nainstalovat na hostitelském PC obecné balíčky pro vývoj aplikací (C/C++) build-essential a subversion (obsahující zdroje OpenWRT).

```
sudo apt-get update
sudo apt-get install build-essential subversion
```

Dále je nutné nainstalovat balíčky uvedené v závislostech OpenWRT

```
sudo sudo apt-get install asciidoc binutils bzip2 fastjar flex g++ gcc autoconf gawk bison libgtk2.0-dev intltool jikes zlib1g-dev make libncurses5-dev libssl-dev patch perl-modules rsync ruby sdcc unzip wget sdcc-nf gettext xsltproc zlib1g-dev
```

Pomocí subversion následně stáhneme OpenWRT ze zdrojového repositáře (Doporučeno je provádět v domovském adresáři kvůli kratší cestě (pavel@debian:~/)).

```
svn co svn://svn.openwrt.org/openwrt/branches/backfire
```

V adresáři backfire spusťte `make menuconfig` a zde vyberte balíčky a moduly potřebné pro instalaci (dvě zmáčknutí mezerníku pro implementaci do vznikajícího systému). V práci byla použita tato konfigurace

```
target system Atheros -AR71xx/AR913x
subtarget - generic
Target Profile - Atheros (madwifi)
Advenced config option (for developes)
      - use chache
       toolchain options
            - binutils 2.20
            - gcc 4.3.3
Basic system
      -block-mount
      -libgcc
      -libpthread
      -libstdcpp
Kernel modules
      Block devices
            -kmod-scsi-core
      Cryptografic API modules
            -kmod-crypto-core
            -kmod-crypto-aes
            -kmod-crypto-arc4
            -kmod-crypto-core21 (missing)
            -kmod-crypto-md5
```

```
      Fylesystem modules
            -kmod-fs-ext2
            -kmod-fs-ext3
            -kmod-fs-msdos
      USB support
            -kmod-usb-core
            -kmod-usb-ohci
            -kmod-usb-storage
            -kmod-usb2
Wireless Drivers
      -kmod-ath
      -kmod-ath9k
      -kmod-cfg80211
      -kmod-mac8021 (missing!!)
Network Support
      -hostapd-mini
      -hostapd-utils
      -wpad-mini
      -wprobe util
Libraries
      -libncurses
      -libncursessw
      -libpcap
      -libuci-lua
      -libnl
Utilities
      -fdisk
-Build Image
-Build Toolchan
-Build SDK
```

Kompilace se epouští příkazem make –j4 (číslo označuje počet souběžně jdoucích úloh)

Nyní je třeba modifikovat proměnnou PATH pro správnou funkci vytvořeného kompilátoru gcc/g++. Provádí se v souboru `~/.profile` kde přidáte následující řádek

```
export          PATH=/home/pavel/backfire/staging_dir/toolchain-mips_r2_gcc-4.3.3+cs_uClibc-0.9.30.1/usr/mips-openwrt-linux-uclibc/bin:${PATH}
   (PATH=/home/pavel/backfire/staging_dir/toolchain-mips_r2_gcc-4.3.3+cs_uClibc-0.9.30.1/usr/bin:${PATH})
```

V menuconfig byla definována cílová architektura MIPS (s druhým vydáním instrukční sady ISA mips32_r2) s použitou vývojovou knihovnou uClibc. Toolchan má přednastavené cesty ke knihovnám na cílovém systému. Umístění knihoven v OpenWRT odpovídá na hostitelském PC umístění ve staging_dir/target-openwrt-mips-uclibc…

Kompilátor gcc má tvar `mips-openwrt-linux-uclibc-gcc`

Knihovny je třeba definovat v adresáři `lib` a `include` příkazem **-L** a **-I** v odpovídajícím adresáři. Výsledný příkaz pro program využívající knihovnu pcap :

```
./mips-openwrt-linux-gcc main.c -I/home/pavel/backfire/staging_dir/target-mips_r2_uClibc-0.9.30.1/usr/include -L/home/pavel/backfire/staging_dir/target-mips_r2_uClibc-0.9.30.1/usr/lib -lpcap -o main.out
```

Ovládání programu

Program se spouští na přístupovém bodě se šesti parametry: označením rozhraní (např. eth0.1), délkou základního intervalu v sekundách, velikostí okna klouzavého průměru (počet násobků základního intervalu), velikostí okna stochastického oscilátoru (jedna hodnota pro indikátor K druhá pro indikátor D) a s velikostí okna bollinger bands

Např.: `./qubik wlan0 3 5 5 2 5`

Program se ukončuje zadáním písmena "q" a jeho potvrzením entrem

Dále je pro použité (bezdrátové) rozhraní nutno změnit MAC adresu na d8:5d:4c:f7:02:fe, nebo ji změnit před kompilací programu