VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

BEZPEČNÝ TRANSPORT NETFLOW A IPFIX DAT

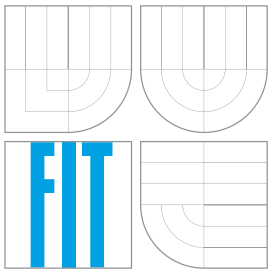
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ADAM ŠTĚPÁNEK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

BEZPEČNÝ TRANSPORT NETFLOW A IPFIX DAT

SECURE TRANSPORT OF NETFLOW AND IPFIX RECORDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ADAM ŠTĚPÁNEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ PODERMAŃSKI

BRNO 2012

Abstrakt

Bakalářská práce se zabývá monitorováním datových sítí na bázi IP flow. V první části představuje architekturu monitorování na bázi NetFlow, popisuje základní pojmy architektury, samotný protokol NetFlow a jeho případné alternativy. Následně jsou zhodnocena slabá místa monitorovacích systémů a je navrženo konceptuální řešení. Toto řešení je implementováno a implementace detailně popsána. Na závěr jsou vysvětleny metody a výsledky testování a diskutují se možnosti rozšíření a optimalizace aplikace.

Abstract

This bachelor thesis deals with an IP flow based data network monitoring system. It presents the architecture of the NetFlow based monitoring, explains the basic terms, the NetFlow protocol and its alternatives. Further, weak spots of the monitoring systems are determined and a conceptual solution is proposed. This solution is implemented and described in detail. Finally, testing methods and results are discussed and the possibilities of further development and optimization are proposed.

Klíčová slova

počítačové sítě, NetFlow, IPFIX, monitorování sítí, sonda, kolektor, Python, SQLite

Keywords

computer networks, NetFlow, IPFIX, network monitoring, exporter, collector, Python, SQLite

Citace

Adam Štěpánek: Bezpečný transport netflow a ipfix dat, bakalářská práce, Brno, FIT VUT v Brně, 2012

Bezpečný transport netflow a ipfix dat

Prohlášení

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

.....
Adam Štěpánek
4. května 2012

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Tomáši Podermański za ochotu a pomoc při tvorbě této bakalářské práce.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Monitorování datových sítí je významná součást celé síťové a internetové infrastruktury. Existuje mnoho rozličných využití. Usnadňuje efektivní plánování budoucího rozšiřování kapacity sítě, uchovávání dlouhodobých statistik síťového provozu[1], detailní sledování uživatelů a služeb a účtování na základě přenesených dat. Dále umožňuje bezpečnostní analýzu v reálném čase a rozpoznání anomálií jako jsou červi a DDoS útoky, snadné plánování a monitorování QoS a z toho plynoucí celkové zvyšování spolehlivosti.

Protože je monitorování sítí velmi důležité, je nutné zajistit spolehlivé získávání a uchovávání informací o síťovém provozu. Tato práce diskutuje problémy a řešení monitorování na bázi IP flow.

Kapitola 2 se zabývá architekturou monitorování na bázi flow, jsou vysvětleny způsoby připojení monitorovacích sond, základní pojmy, následně podrobně popsán protokol NetFlow a jsou zmíněny jeho alternativy. Kapitola 3 diskutuje slabá místa monitorovacích systémů a v kapitole 4 je nastíněno řešení – koncepce spolehlivého transportu. Toto řešení je implementováno a implementace detailně popsána v kapitole 5. Kapitoly 6 a 7 představují průběh testování, jeho výsledky a možnosti rozšíření a optimalizace aplikace. Celá práce je shrnuta v závěrečné kapitole 8, kde je také vysvětlen její přínos.

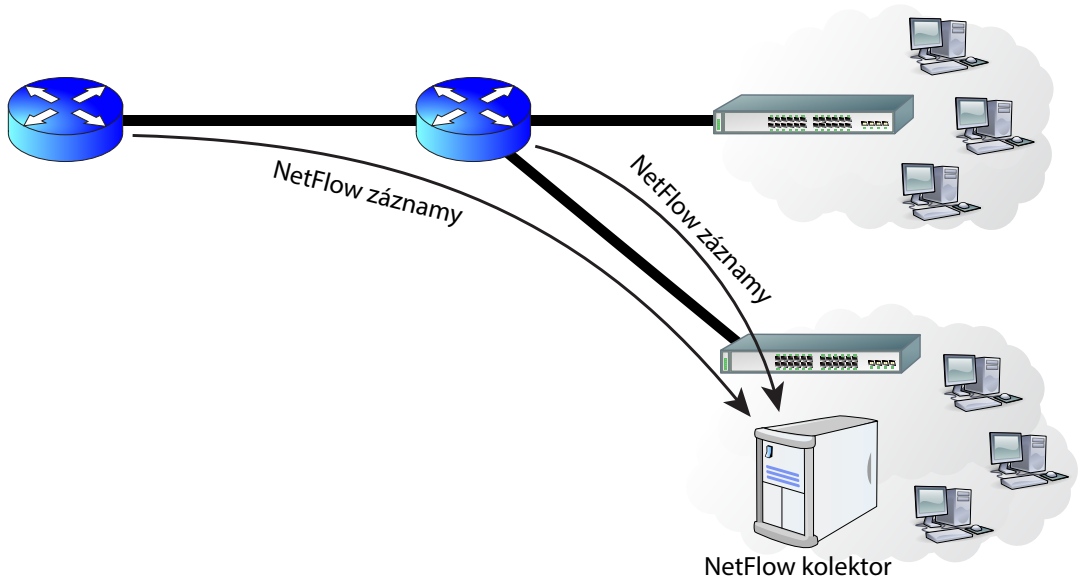
[1] V některých zemích vyžadováno legislativou.

Kapitola 2

Monitorování datových sítí

2.1 Architektura monitorování sítí na bázi flow

Architektura monitorování sítí na bázi NetFlow pracuje se dvěma základními pojmy. Prvním z nich je exportér, tj. obecně zařízení, které poslouchá síťový provoz a generuje záznamy o monitorované síti, a kolektor, což je zařízení, kam jsou tyto záznamy posílány. Zde mohou být podle požadavků uchovány a zpracovány pro snadnou diagnostiku.



Obrázek 2.1: Architektura NetFlow využívající směrovačů

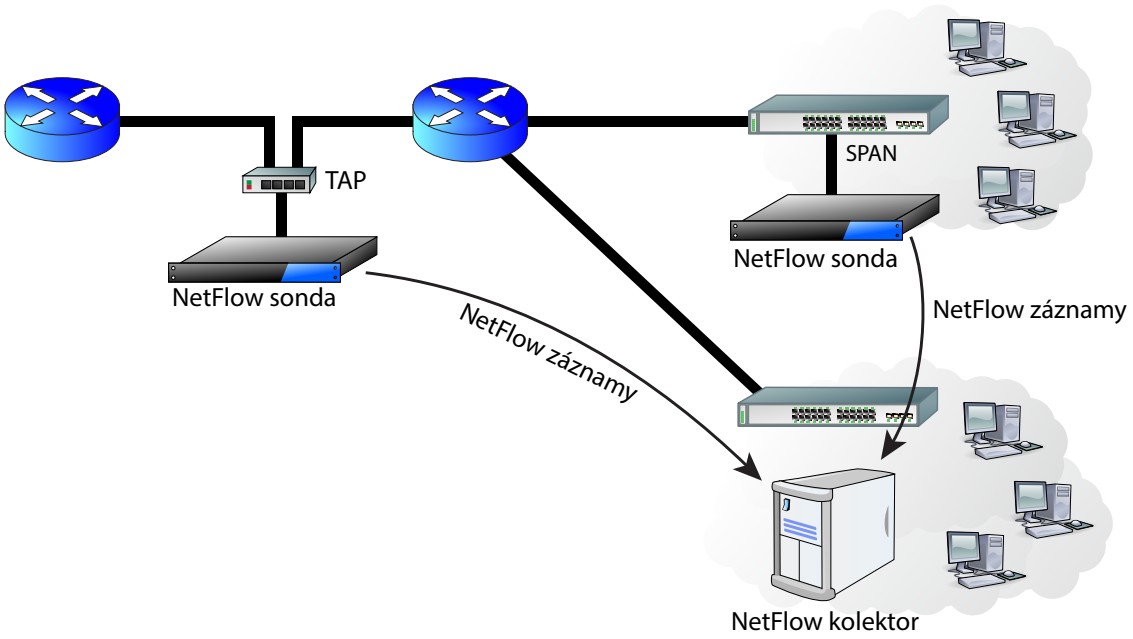
Existuje více možností, jak monitorovat provoz na síti. První řešení využívá pro svou činnost přímo směrovačů, popř. rozšiřujících karet, které lze ke směrovači připojit. Zde jsou pakety zachytávány a přeposílány na kolektor, jak je zobrazeno na obrázku 2.1. Nevýhodou je velká náročnost na hardware a tedy i vyšší pořizovací cena takových zařízení. Většinou se tedy používá pouze ve specializovaných aplikacích či rozsáhlejších sítích. Další nevýhodou je nepřesnost ve statistikách způsobená vzorkováním příchozích dat, tedy zpracováváním pouze každého n-tého paketu. Zatížení směrovačů v rozsáhlých a intenzivně využívaných

sítích totiž může být velmi vysoké a směrovači už nemusí zbývat prostředky pro zpracování NetFlow dat. Tímto způsobem pracuje většina směrovačů, s výjimkou těch nejdražších a nejvýkonnějších modelů. Také při použití rozšiřující karty není vzorkování obvykle nutné.

Další možností je použití sond. Můžeme využít buď zařízení k tomuto účelu přímo určená nebo počítač s nainstalovaným softwarem pro zpracování NetFlow dat. Jednoúčelová zařízení mají obvykle specializovaný hardware, a tudíž jsou velmi výkonná. Software je dodáván přímo výrobcem sondy a není možné do něj zpravidla jednoduše zasahovat. Cena těchto zařízení bývá různá. Liší se především podle datové propustnosti a rychlosti portů, které obsahuje. Příkladem takové sondy může být FlowMon brněnské firmy INVEA-TECH [14].

Software běžící na počítači nedosahuje takové výkonnosti a nedokáže zpracovat tak velké objemy toků jako drahá jednoúčelová zařízení, zato je možné si vytvořit vlastní řešení, nejčastěji jako program běžící na operačním systému Linux, FreeBSD apod. Tímto způsobem můžeme vytvořit velice variabilní řešení. Příklady takových aplikací mohou být např. nProbe [20], fProbe [1] nebo NDSAD [19].

Sondy jsou k lince připojeny pomocí TAP nebo SPAN portů. Situaci zachycuje obrázek 2.2. Podrobněji jsou tyto způsoby popsány v kapitole 2.2.



Obrázek 2.2: Architektura NetFlow využívající pasivní sondy

Kolektorem bývá nejčastěji server s velkou úložnou kapacitou, který slouží jako úložiště NetFlow záznamů. Musí zde běžet specializovaný software, který je schopný NetFlow záznamy přijímat, ukládat, ve vhodné formě je zobrazovat administrátorovi, popř. i filtrovat podle požadovaných kritérií. Příkladem takového programu je open source NFDUMP [11] a komerční aplikace Flow Inspector [3] nebo NetFlow Tracker [9].

2.2 Způsoby připojení monitorovacích sond

První způsob připojení je možný skrze port TAP (Test Access Port), který se nachází v zařízeních zvaných network taps. Jedná se zpravidla o pasivní zařízení, která zachycují tok dat kdekoliv mezi dvěma síťovými zařízeními (např. směrovači). Data se v network tap duplikují a bez zpoždění odesílají výstupním portem a zároveň TAP portem na analýzu. Vstupní i výstupní tok mají oddělené linky pro duplikovaná data. Výhodou tohoto řešení je zachycení všech paketů i v případě plného zatížení linky a garance doručení dat v pořadí, v jakém byla odeslána. Další výhodou je nízká náchylnost k chybám z důvodu pasivity celého zařízení. Pokud se jedná o network tap závislý na napájení, je konstruován tak, aby v případě výpadku napájení nedošlo k přerušení přeposílání dat. Analyzovaná data obsahují kompletní provoz odpovídající toku na lince, včetně ISO/OSI vrstev 1 a 2, VLAN tagů a poškozených rámců [8].

Další možností připojení jsou SPAN (Switch Port Analyzer) porty, speciální porty využívané na přepínačích, do kterých se připojuje monitorovací zařízení (sonda). Přepínač duplikuje provoz a odesílá jej přes SPAN port. Takové řešení však má několik nevýhod. Časy paketů odeslaných do skutečné destinace a do monitorovacího zařízení si přesně neodpovídají kvůli použití vyrovnávacích pamětí. Pokud dojde k vysokému vytížení přepínače, může dojít k zahazování paketů. Pro použití SPAN portů je nutná konfigurace přepínače. Ne všechna data jsou skutečně poslána přes SPAN port – jedná se především o poškozené pakety, které mohou být způsobeny vadnou síťovou kartou (NIC) a které přepínač zahazuje [18]. V některých implementacích mohou být ignorovány VLAN tagy, takže je obtížné monitorovat problémy týkající se virtuálních LAN sítí. Některé informace z linkové ISO/OSI vrstvy mohou být modifikovány (např. cílová MAC adresa). V neposlední řadě není zaručeno správné pořadí doručení paketů [8].

Všechny tyto nevýhody jsou důvodem, proč nelze monitorování skrze SPAN porty nazývat skutečným pasivním monitorováním dat. V některých případech je použití SPAN portů opodstatněné, např. pokud se jedná o monitorování pouze malých objemů dat, kdy nemůže dojít k zahazování paketů v důsledku vysokého vytížení přepínače. Dále je toto řešení velmi vhodné, pokud chceme monitorovat více linek najednou. Přístup s použitím TAP portů by byl komplikovaný, protože by se na každou linku musel umístit network tap a ke každému network tap připojit exportér. Použití SPAN portu situaci značně zjednodušuje, kdy v přepínači pouze nastavíme linky, které potřebujeme monitorovat a připojíme jeden exportér.

Obě řešení mají své výhody a nevýhody. Volbu ovlivňuje spousta faktorů a záleží na konkrétní situaci nebo prostředí, ve kterém chceme monitorování použít.

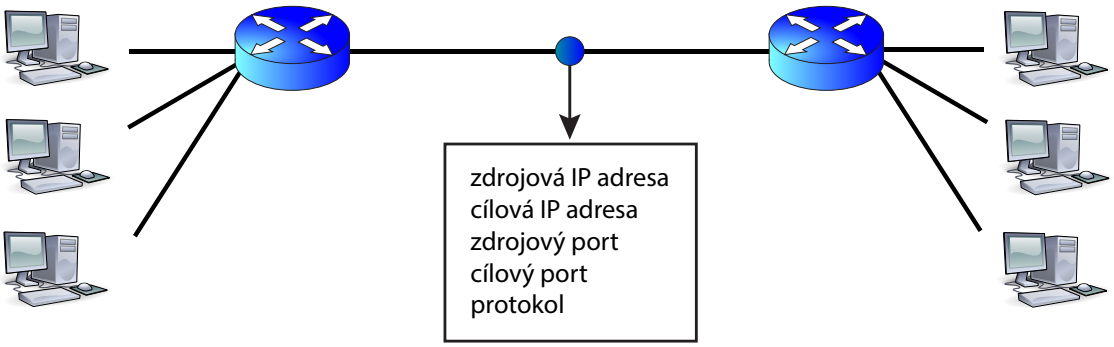
2.3 IP tok

IP tok je naprosto základní pojem celé architektury NetFlow. Na základě IP toků se v kolektoru generují statistiky provozu, a proto je nutné se s tímto pojmem blíže seznámit.

Tradičně se za IP tok považují pakety, které mají společnou pětici atributů (obrázek 2.3):

- IP zdrojová adresa
- IP cílová adresa
- zdrojový port TCP nebo UDP

- cílový port TCP nebo UDP
- číslo protokolu na 3. vrstvě ISO/OSI modelu



Obrázek 2.3: Tradiční pětice atributů definující IP tok

Někdy se jako atributy přidávají ještě typ služby (type of service) a vstupní rozhraní (tzv. ifIndex). Situace ale dnes není tak jednoznačná a již např. RFC 3954 [7] z roku 2004 mluví o IP toku více obecně. Definuje jej jako soubor paketů procházející přes určité kontrolní místo v síti za určitý čas. Všechny takové pakety mají společné vlastnosti odvozené z dat v paketu a podle způsobu, jakým je paket zpracován v kontrolním místě. S příchodem nových verzí protokolu NetFlow (kapitola 2.4) je totiž možné konfigurovat atributy IP toku a nadefinovat je tak podle potřeby.

Ještě obecněji pak můžeme nazvat IP tok jako síťový. Definice síťových toků lze najít v různých pojetích, např. v [2], [24] nebo v [23].

Atributy jsou pro jeden tok společné a neměnné. Důležité je upozornit, že tok je jednosměrný. Pokud komunikuje zařízení A s IP adresou X na portu XX se zařízením B s IP adresou Y na portu YY přes protokol TCP a naopak, jedná se o dvojici toků – od zařízení A a k zařízení A. Při načítání webové stránky přes jedno TCP spojení se tedy jedná o dvojici toků.

Na základě IP toků se generují NetFlow záznamy, které se odesílají na kolektor. Každý záznam obsahuje jednak klíčové položky, tedy atributy podle kterých tok identifikujeme (zdrojová a cílová IP, zdrojový a cílový port atd.), a dále agregační položky, tedy informace, které se kumulují z více paketů. Může se jednat o informace o začátku přenosu, konci přenosu, počtu přenesených paketů, celkové množství dat a další údaje. Podrobněji je NetFlow záznam popsán v kapitole 2.5.

K odeslání záznamu o toku dochází v těchto případech: [7]

- Exportér dokáže detekovat konec toku, např. FIN nebo RST bitem u TCP spojení.
- Pokud je tok dlouhou dobu neaktivní, tzn. daným místem v síti neprošel žádný paket patřící do tohoto toku po určitou dobu (tzv. neaktivní timeout).
- U dlouhodobých toků exportér odesílá záznam v pravidelných intervalech (tzv. aktivní timeout).
- Při dosažení vnitřních omezení exportéru, především zaplnění dočasné paměti.

2.4 Přehled verzí protokolu NetFlow

Protokol NetFlow je síťový protokol vyvinutý firmou Cisco Systems pro monitorování přenosů dat po IP sítích. Původně byl zamýšlen pouze jako služba ke směrovačům firmy Cisco, později se však rozšířil na směrovače dalších značek a do monitorovacích sond. Jeho první verze se netěšila přílišné popularitě. Verze 2, 3 a 4 se na trh vůbec nedostaly, jednalo se pouze o interní vydání. Přelomovou se stala verze 5, která je dodnes (2012) zřejmě nejpoužívanější v rodině protokolů NetFlow. Největší nedostatky leží v absenci IPv6 a celkové nemožnosti jednoduše protokol rozšířit bez vydávání nové verze. Většina dnešních směrovačů Cisco, na kterých běží IOS 11.1 nebo vyšší, verzi 5 podporuje. Stejně jako verzemi 2, 3 a 4, ani verzemi 6, 7 a 8 se není nutné nějak hluboce zabývat. Oproti verzi 5 nepřináší žádné velké výhody a většinou již nejsou podporované nebo je jejich rozšíření v reálném nasazení mizivé.

Další přelomovou verzí se stala až verze 9. Největším přínosem je použití systému šablon, díky kterému bylo možné přidat do protokolu mnoho nových vlastností, včetně podpory IPv6. Kromě IPv6 podporuje multicast toky, IPsec nebo Multi Protocol Label Switching (MPLS). U Cisco směrovačů byla přidána podpora verze 9 v IOS 12.4. Velká většina dnešních směrovačů tak verzi 9 podporuje. Protože se tato bakalářská práce týká převážně NetFlow verze 9, bude protokol popsán detailněji v samostatné podkapitole 2.5.

Rozšířením verze 9 vznikl Flexible NetFlow. Největším přínosem je možnost explicitního určení exportovaných dat a možnost určení atributů, podle kterých bude definován a zachycován IP tok [5].

V roce 2006 uveřejněný protokol IP Flow Information Export (IPFIX) vychází z NetFlow verze 9 a jeho cílem je snaha o standardizaci a oproštění se od závislosti na proprietárním protokolu NetFlow. Za jeho vydáním stojí celosvětová internetová organizace Internet Engineering Task Force (IETF). Standard je popsán v experimentálním RFC 3917 [23]. IPFIX popisuje, jakým způsobem jsou data zachycována na sondě a následně odesílána do kolektoru [13]. Mezi výrobce podporující IPFIX patří Juniper, Avaya, SonicWALL nebo nBox.

Tato práce se zabývá především NetFlow verzí 9, protože je oproti IPFIX, ačkoliv se jedná o standard, rozšířenější a mezi výrobci zařízení více podporovaná. Protože však IPFIX z verze 9 vychází, existuje mezi nimi mnoho podobností a velká část principů se dá aplikovat jak na verzi 9, tak na IPFIX.

2.5 NetFlow verze 9

Novinkou verze 9 oproti starším verzím je systém šablon. Tento systém s sebou přináší několik výhod. Nová pole mohou být přidávána bez toho, aniž by se zasahovalo do samotné struktury protokolu. Šablony tak představují do budoucna snadnější modifikaci a přidávání nových rozšíření. Pokud bude sonda posílat do kolektoru záznamy s poli, kterým kolektor nerozumí, stále je možné tyto pole ignorovat a zpracovat zbytek záznamu. Také je možné explicitně na sondě určovat, která pole nás zajímají a tedy která chceme zasílat. Zasílání pouze důležitých informací tak může významným způsobem zredukovat množství přenášených dat na lince ke kolektoru.

NetFlow záznamy jsou typicky odesílány kvůli efektivitě ve formě UDP paketů. Protože je ale verze 9 protokolově transparentní, lze použít jakýkoliv jiný protokol.

Kompletní specifikace protokolu NetFlow verze 9 se nachází v RFC 3954 [7].

2.5.1 Terminologie

NetFlow technologie používá několik pojmů, které je nutné před vlastním výkladem struktury pakety, vysvětlit.

- *Export paket (angl. export packet)* – paket, který vytváří zařízení (směrovač, sonda), a který je odesílán na další zařízení (kolektor).
- *Hlavička paketu (angl. packet header)* – první (povinná) část export paketu. Obsahuje základní informace jako verze NetFlow, počet záznamů obsažených v paketu aj.
- *FlowSet* – obecný pojem pro soubor záznamů, které následují za hlavičkou. Existují 2 typy FlowSet: *šablona* a *data*. Export paket obsahuje jednu nebo více FlowSet částí a *FlowSet šablona* i *FlowSet data* mohou být obsaženy v rámci jednoho paketu.
- *FlowSet šablona (angl. template FlowSet)* – kolekce jednoho nebo více *šablonových záznamů*, které se nacházejí v rámci jednoho paketu.
- *Šablonový záznam (angl. template record)* – určuje strukturu dat, která budou následovat, a to v současném paketu i v budoucích paketech. Důležité je, že *šablonový záznam* nutně neříká nic o tom, že struktura dat, kterou určují, se musí nutně nacházet ve stejném paketu. Kolektor se musí postarat o ukládání šablon do své dočasné paměti a následně analyzovat a zpracovat datové záznamy podle těchto uložených šablon.
- *ID šablony (angl. template ID)* – jedinečné číslo, které identifikuje šablonu a odlišuje ji od ostatních šablon generovaných stejným exportérem. Protože jsou data do kolektorů často posílána z více exportérů a protože je jedinečnost ID zaručena pouze v rámci jednoho exportéru, kolektor by si měl spolu s ID ukládat i adresu zařízení.
- *FlowSet data (angl. data FlowSet)* – kolekce jednoho nebo více *datových záznamů*, které se nacházejí v rámci jednoho paketu.
- *Datový záznam (angl. data record)* – poskytuje informace o toku, který proběhl nebo probíhá na zařízení, které export paket vytvořilo. Každý *datový záznam* v sobě mj. obsahuje odkaz na ID šablony, podle které byl tento *datový záznam* vytvořen.
- *Šablona rozšířeného záznamu (angl. options template record)* – speciální typ *šablonového záznamu*, který určuje strukturu dat v *rozšířeném datovém záznamu*.
- *Rozšířený datový záznam (angl. options data record)* – speciální typ *datového záznamu*, který poskytuje informace o samotném NetFlow procesu.

2.5.2 Struktura paketu

NetFlow paket verze 9 se skládá z hlavičky a minimálně jednoho FlowSet. *FlowSet šablony* i *FlowSet data* mohou být umístěny libovolně v rámci jednoho paketu, jak ukazuje tabulka 2.1.

Export paket může obsahovat *FlowSet šablonu/šablony* i *FlowSet data*, pouze *FlowSet data* (v reálném provozu je většina paketů tohoto typu) nebo pouze *FlowSet šablonu/šablony*.

Hlavička paketu	FlowSet šablona	FlowSet data	FlowSet data	...	FlowSet šablona	FlowSet data

Tabulka 2.1: Struktura NetFlow paketu verze 9

Struktura hlavičky

Hlavička verze 9 (2.2) je založena na verzi 5 a příliš se od starších verzí neliší.

- *Version Number* – pro verzi 9 se v tabulce nachází hodnota 0x00009.
- *Count* – celkový počet FlowSet záznamů v paketu.
- *System Uptime* – Doba běhu zařízení od spuštění v miliseknudách.
- *UNIX Seconds* – Počet sekund od UTC v roce 1970.
- *Sequence Number* – Navyšující se čítač pro všechny pakety odeslané exportérem. Podle této hodnoty může kolektor poznat, zda nedošlo ke ztrátě paketu.
- *Source ID* – 32 bitová jedinečná hodnota pro všechny toky odeslané z exportéru. Typicky se odvozuje od IP adresy rozhraní s nejnižší hodnotou, ale lze ji nastavit i manuálně.

0	1	2	3
Version Number		Count	
System Uptime			
UNIX seconds			
Sequence Number			
Source ID			

Tabulka 2.2: Struktura hlavičky NetFlow záznamu

Struktura FlowSet šablony

- *FlowSet ID* – Hodnota slouží pro odlišení FlowSet šablon od FlowSet dat. Hodnota 0 je rezervována pro šablony. Pro data se využívají kladné hodnoty větší než 255.
- *Length* – Celková velikost daného FlowSet.
- *Template ID* – Každému nově vygenerovanému šablonovému záznamu je přiřazeno jedinečné číslo – Template ID. Šablony, které definují data, mají ID větší než 255. Hodnoty 0 až 255 jsou rezervovány pro FlowSet ID.
- *Field Count* – Počet polí v tomto šablonovém záznamu. Tato hodnota slouží kolektoru k určení konce šablonového záznamu a začátku následujícího (pokud existuje).

- *Field Type* – Číselná hodnota určující typ pole (podrobnější informace v [7]).
- *Field Length* – Velikost v bytech korespondujícího pole *Field Type*.

0	1	2	3
FlowSet ID = 0		Length	
Template ID 256		Field Count	
Record 1 – Field Value 1		Record 1 – Field Value 2	
Record 1 – Field Value 3		...	
...		...	
Field Type N		Field Length N	
Template ID 257		Field Count	
Field Type 1		Field Length 1	
Field Type 2		Field Length 2	
...		...	
Field Type M		Field Length M	
...		...	
Template ID K		Field Count	
...		...	

Tabulka 2.3: Struktura FlowSet šablony

Struktura FlowSet dat

- *FlowSet ID (= Template ID* – Každý datový FlowSet má své ID. Toto ID odpovídá některému ze šablonových ID.
- *Length* – Celková velikost datového FlowSet včetně pole *Padding* (pokud existuje).
- *Record N - Field Value N* – Soubor datových záznamů, jejichž význam je určen korespondujícími šablonovými záznamy.
- *Padding* – Pokud je to třeba, jsou vloženy 2 nulové bajty, aby byl FlowSet zarovnaný na násobek 4 bajtů.

Šablona rozšířeného záznamu a rozšířený datový záznam

Šablona rozšířeného záznamu a rozšířený datový záznam k sobě mají stejný vztah jako FlowSet šablona k FlowSet datům. Rozšířený datový záznam neposkytuje informace o tocích, ale o vlastním NetFlow procesu zachytávání těchto toků. Jako příklad lze uvést informaci o tom, zda je podporováno vzorkování, popř. jaká je vzorkovací frekvence nebo jaká je použita metoda vzorkování.

0	1	2	3
FlowSet ID = Template ID		Length	
Record 1 – Field Value 1		Record 1 – Field Value 2	
Record 1 – Field Value 3		...	
Record 2 – Field Value 1		Record 2 – Field Value 2	
Record 2 – Field Value 3		...	
Record 3 – Field Value 1		...	
...		Padding	

Tabulka 2.4: Struktura FlowSet dat

2.6 Alternativy protokolu NetFlow

2.6.1 sFlow

Alternativou k NetFlow může být např. sFlow. Konceptuálně jsou NetFlow a sFlow velmi podobné technologie. Implementační zpracování se však liší.

sFlow ve své terminologii nazývá exportéry jako agenty. Dokument specifikující sFlow verze 5 popisuje především mechanismus vzorkování použitý v agentech, modul MIB sloužící pro vzdálenou správu a konfiguraci sFlow agentů a strukturu sFlow datagramu.

U sFlow začíná celý proces zachycením paketu ve směrovači/přepínači. Agentovi je předána hlavička paketu, informace o vstupním a výstupním rozhraní a vzorkovací parametry. Agent přidá další relevantní informace, včetně hodnot čítačů na rozhraních a předá paket k odeslání. Ten je ihned přenesen do kolektoru. U sFlow tedy neprobíhá agregace více paketů do jednoho záznamu, ale pro každý zachycený paket je vytvořen vlastní záznam.

Na vzorkování dává sFlow poměrně velký důraz. Používají se dva typy vzorkování, které se nevylučují, ale navzájem se doplňují. Prvním z nich je vzorkování pro výběr paketů, které se budou analyzovat. Toto vzorkování probíhá podle algoritmu, který má díky své náhodnosti zajistit pro každý paket stejnou šanci analyzování. Obecně se analyzuje 1 z N paketů. Ani při vhodně zvolené frekvenci vzorkování není dosažena 100% přesnost, ale lze dosáhnout výsledků použitelných pro zamýšlený účel. Pokud je číslo N příliš vysoké, snižuje se přesnost získaných údajů a ve výsledných statických datech mohou chybět velké části provozu na monitorované lince.

Druhý typ vzorkování určuje frekvenci odesílání záznamů o tocích do kolektoru. Celý mechanismus je podrobně popsán v [21].

sFlow byl původně vyvinut firmou inMon. Verze 2 i verze 4, popsaná v RFC 3176, jsou v současné době zastaralé a všechny autoritativní informace o nejnovější páté vývojové verzi poskytuje konsorcium sFlow.org skrze své stejnojmenné webové stránky. Jedná se o otevřený standard a podporuje jej i firma HP.

2.6.2 jFlow, AppFlow a další

Další alternativou může být jFlow [16]. Jedná se o protokol velmi podobný NetFlow. Využívá jej ve svých směrovačích a přepínačích např. firma Juniper Networks.

Stejně jako jFlow, i AppFlow [6], NetStream [12] a další, mají jedno společné. Jedná se

o deriváty protokolu NetFlow, resp. IPFIX, které poskytují velmi podobné nebo dokonce stejné služby, avšak každá společnost používá pro svůj derivát jiné označení.

Kapitola 3

Slabá místa monitorovacích systémů

Nic není na světě dokonalé, ani zařízení a technologie používané pro monitorování sítí. Vždy se musí počítat se situací, že v některé části monitorovacího systému nastane chyba, popř. dojde k jeho úplné nefunkčnosti. V takových případech bychom potřebovali informace o monitorované síti velmi nutně pro zjištění příčin nefunkčnosti a přitom je kvůli výpadku nemáme. Dochází tedy k problému, který je nutné řešit.

Pro následující analýzu problému slabých míst systému budeme vycházet z obrázku 2.2.

3.1 Network tap

Pokud používáme monitorování skrze zařízení network tap, může se stát, že toto zařízení přestane fungovat. Abychom zabránili přerušení monitorování, můžeme např. umístit 2 network tapy za sebou. Pokud přestane jeden z nich duplikovat data, může jej nahradit druhý. Problém by nastal v případě, že by network tap přestal data odesílat úplně, tedy nefungovalo by ani duplikování, ale ani odesílání skrze výstupní port. V takovém případě by řešením byla pouze výměna vadného kusu. Pokud bychom takovou chybu brali v potaz, je použití duplicitních tapů kontraproduktivní, protože existuje dvojnásobné šance této chyby.

Vzhledem k tomu, že network tap nebývá příliš složité zařízení, šance, že dojde k chybě, nebývají tak vysoké jako u složitějších zařízení. Využívají se dokonce network tapy bez napájení (např. pro optické linky), u kterých je riziko poruchy velice nízké.

Ani při normálním běhu obou zařízení není situace jednoduchá. Do exportéru tečou dva proudy dat, z nichž exportér vybere pouze jeden, který se použije. Musí tedy existovat určitý alogirtmus pro výběr porovnávající oba proudy dat a celý systém se tím dále komplikuje.

Z těchto předpokladů nám vyplývá, že použití duplicitních tapů není vhodné. Pokud dojde k chybě, je nutná výměna zařízení.

3.2 SPAN port

U přepínače může dojít k chybě SPAN portu. Nezáleží na tom, zda přestane fungovat pouze samotný SPAN port nebo více části přepínače, či dokonce celý přepínač. Ve všech případech je nutná oprava zařízení a v produkčním prostředí samozřejmě okamžitá výměna za bezvadný kus.

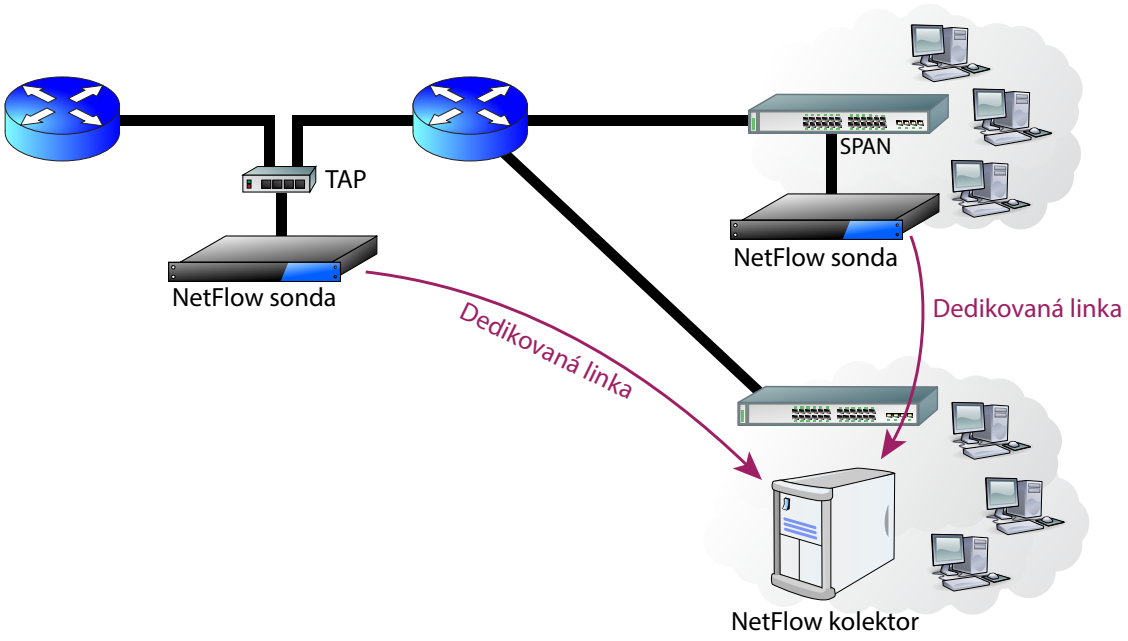
3.3 NetFlow sonda

U hardwarové NetFlow sondy je situace obdobná jako u chyby SPAN portu. V případě poruchy je nutná oprava zařízení spojená s okamžitou výměnou za bezvadný kus. Výrobce v takovém případě může nabízet zapůjčení náhradního zařízení na dobu opravy původního, což je ale většinou promítnuto do vyšší ceny, ať už za samotné zařízení, rozšířenou záruku nebo ve formě jiného poplatku.

U softwarové sondy je situace odlišná. Pokud se chyba objeví u softwarové části (aplikace), je nutné kontaktovat dodavatele aplikace. Toho informujeme o problémech a závadách na aplikaci a obvykle od něj požadujeme rychlé zjištění příčin, nápravu a dodání nové verze softwaru. Pokud přestane fungovat hardwarová část (počítač), je nutná výměna nefunkční části nebo celého kusu.

3.4 Linka přeposílající data ke kolektoru

Linku přeposílající data ke kolektoru najdeme v NetFlow architektuře využívající pasivní sondy, jak je vidět na obrázku 2.2. Samotná dedikovaná linka je pak vyznačena níže na obrázku 3.1.



Obrázek 3.1: Dedikovaná linka v architektuře NetFlow využívající pasivní sondy

Je nutné si uvědomit, že v reálném prostředí mohou být exportér a kolektor od sebe umístěny na velkou vzdálenost. Pokud by se jednalo o dvě datová centra, může se tato vzdálenost pohybovat v kilometrech, ale i v desítkách kilometrů. Řešením může být použití duplicitních kabelů. Takové řešení by stálo navíc finanční prostředky za samotnou kabeláž a v případě velkých vzdáleností již nemusí jít o zanedbatelné částky. Dále pokud existuje trasa, kudy vedl původní kabel, lze předpokládat, že i pro nový kabel by se tato trasa použila.

V případě události, která by zapříčinila poškození jednoho kabelu (přírodní katastrofa, překopnutí kabelu atd.), existuje velká pravděpodobnost, že by mohlo dojít i k poškození druhého.

Řešení posílající data po duplicitních linkách skrze různé geografické cesty je ještě méně vhodné řešení. V první řadě, i pokud bude mezi oběma kabely dostatečná vzdálenost mezi koncovými body, na začátku i na konci trasy musí stejně dojít téměř k jejich sloučení do jediného svazku a dostali bychom se k předešlému problému. V druhé řadě vytváření další samostatné cesty pro kabel v případě velkých vzdáleností je finančně náročné.

Celkově je řešení využívající dedikovaný fyzický spoj finančně velmi nákladné a používá se spíše na menší vzdálenosti. Běžným řešením je využití existující infrastruktury (např. sítě Internet). Odpadají tak vysoké náklady na budování nové linky a zároveň již bývá vyřešen problém redundance. Data mohou putovat z exportéru do kolektoru po různých cestách skrze více směrovačů díky směrovacím protokolům a pokud dojde k výpadku jedné cesty, použije se jiná. V síti Internet je nevýhodou zasílání citlivých dat skrze veřejně dostupný kanál, ale řešení takového problému není cílem této bakalářské práce.

Důležité je, že v obou případech, při použití dedikovaného fyzického spoje i při použití existující infrastruktury, může dojít ke kolapsu. Velmi výhodným řešením je pak software, který zajistí ukládání záznamů o tocích do paměti a v momentě, kdy dojde k opětovnému zprovoznění cesty od exportéru ke kolektoru, začne odesílat záznamy.

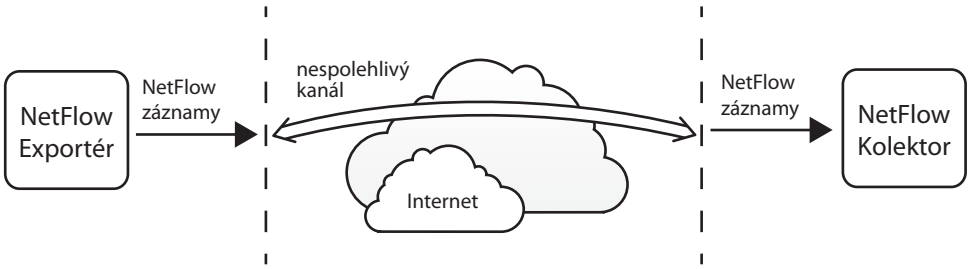
Protože NetFlow používá ve většině případech pro přenos záznamů protokol UDP, může také dojít ke ztrátě paketů např. vlivem velkého vytížení kolektoru. Exportér po odeslání záznamu jej ze své paměti obvykle maže a protože ani nemá prostředky, jak zjistit, že ke ztrátě paketu došlo, informace o toku je nenávratně ztracena.

Oběma těmto případům, tedy úplné nefunkčnosti spojení i případné náhodné ztrátě paketů, se věnují další kapitoly.

Kapitola 4

Koncepce spolehlivého transportu

Standardní situace přenosu NetFlow záznamů z exportéru do kolektoru je koncepčně zobrazena na obrázku 4.1. Exportér vytvoří záznam, který po nespolehlivém kanálu putuje do kolektoru. Tento kanál může být vytvořen jak na dedikovaném fyzickém spoji, tak na jakékoliv jiné infrstruktuře. Často se jedná o síť Internet, kde jsou data posílána protokolem IP, který standardně žádné služby, a tedy ani spolehlivé doručení dat, negarantuje. Pracuje na principu *best effort*, tedy snaží se doručit data příjemci s největším úsilím podle aktuálního vytížení sítě.

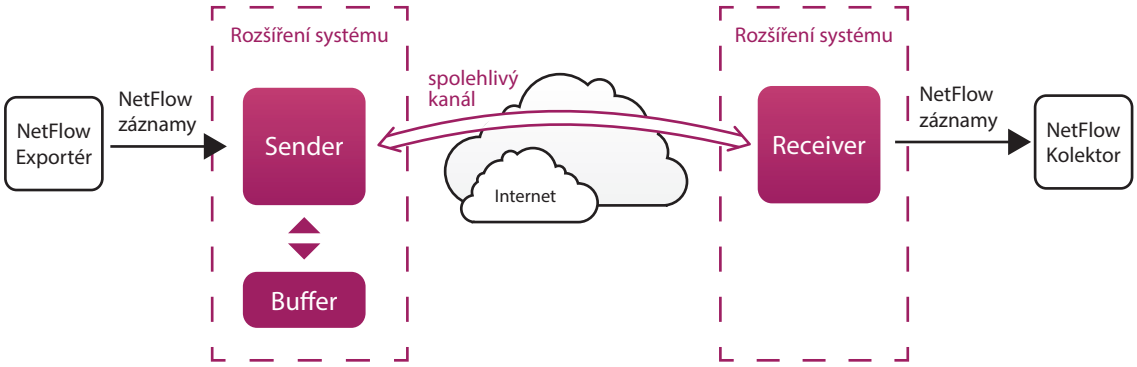


Obrázek 4.1: Tradiční řešení s využitím nespolehlivého kanálu

Řešení navrhované v této bakalářské práci přidává do tradiční architektury dva prvky, jak lze vidět na obrázku 4.2. Prvním z nich je Sender, který přijímá záznamy z NetFlow exportéru. Tyto záznamy následně posílá po spolehlivém kanálu do druhého prvku, Receiveru. Ten přijaté data zpracuje a vytvoří z nich původní NetFlow záznam, který předá kolektoru.

Není nutné, aby prvky Sender a Receiver byly samostatné jednotky. Sender může být součástí stejného zařízení jako exportér, stejně tak Receiver může být součástí zařízení kolektoru.

Kromě prvků Sender a Receiver zůstává celý systém stejný a nemusí se provádět žádné další změny. Především médium, nad kterým je vytvořen spolehlivý kanál, je naprosto stejné, jako bylo použito v tradiční architektuře pro nespolehlivý kanál. Onen abstraktní spolehlivý kanál je vytvořen vhodnými technikami komunikace a zapouzdřování dat Senderu a Receiveru. Lze tedy vidět, že se konceptuálně jedná o neinvazivní řešení, kde se nemusí příliš zasahovat do existujícího systému.



Obrázek 4.2: Navrhované řešení využívající spolehlivý kanál

4.1 Sender

Rozsah činnosti Senderu bychom mohli shrnout do tří hlavních bodů. Prvním úkolem je přijatý NetFlow záznam zapouzdřit do vhodného protokolu. Tento postup je nutný, protože záznamy bývají obvykle zasílány protokolem UDP, který nám neumožňuje nikterak zjistit stav doručení na příjimací straně.

Druhým úkolem je odesílání záznamů, přijetí potvrzování a celkově komunikace s Receiverem, včetně zjišťování stavu linky.

Posledním úkolem je řazení záznamů do fronty a uchovávání v paměti, ať už se jedná o operační paměť nebo pevný disk. Použití operační paměti je vhodné z důvodu vyšší výkonnosti a rychlosti, ale existuje riziko ztráty dat při pádu aplikace nebo operačního systému. Naopak při použití pevného disku je výkonnost nižší, ale riziko ztráty dat při pádu aplikace nebo operačního systému se výrazně snižuje.

4.2 Receiver

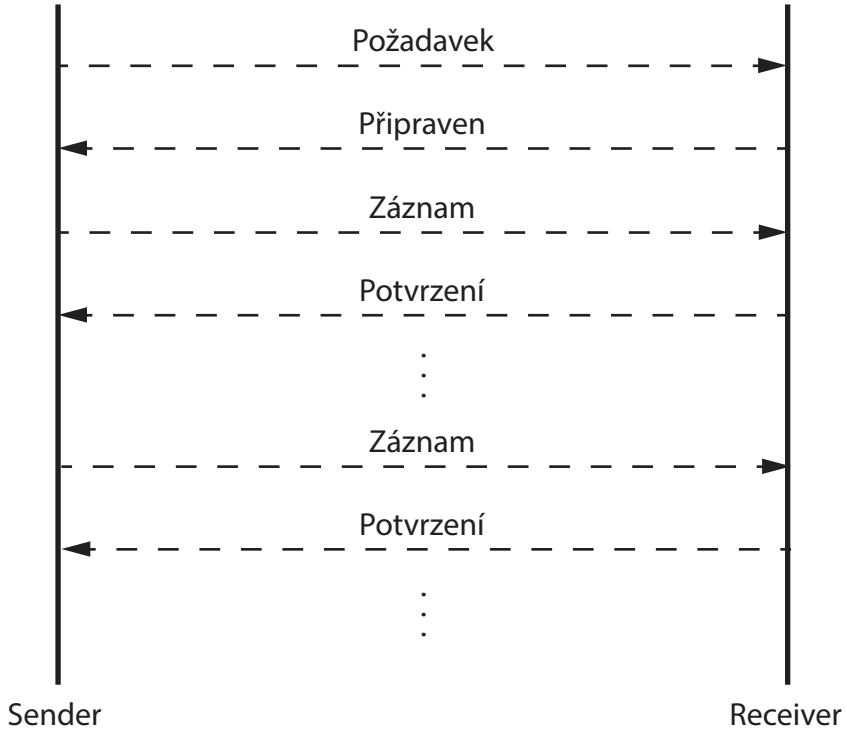
Receiver zahrnuje dva hlavní okruhy činnosti. Za prvé, přijímá data, potvrzuje jejich přijetí a celkově komunikuje se Senderem. Za druhé, odstraňuje zapouzdřovací vrstvu a vytváří původní NetFlow záznam, který předává do kolektoru.

V reálném nasazení se počítá se situací, kdy budou Receiver a kolektor přímo součástí jednoho zařízení nebo jejich propojení bude provedeno spolehlivým způsobem (např. přímé spojení síťovým kabelem). Receiver tedy neobsahuje žádný buffer, protože v této části transportu již nemůže dojít ke ztrátám záznamů z důvodu výpadku linky. Data jsou ukládána pouze do operační paměti a ihned po předání je Receiver odstraňuje.

4.3 Průběh komunikace

Na úplném počátku komunikace odešle Sender dotaz, zda je Receiver připraven přijímat data. Receiver příjme dotaz a odešle potvrzení Senderu. Ten může začít odesílat data, protože ví, že je Receiver připraven je přijímat a Receiver ví, že Sender má připravená data, proto očekává jejich příjem.

V další fázi dochází k odesílání paketů a potvrzování přijetí. Celý proces průběhu komunikace je zachycen na obrázku 4.3.



Obrázek 4.3: Proces průběhu komunikace

Během komunikace je nutné reagovat na chyby. Může dojít ke ztrátám datových paketů i ztrátám potvrzení, proto je nutné zavést systém číslování paketů. Číslo je s každým paketem navyšováno a příjemce jednoduše pozná, pokud neodpovídá očekávanému. Také se systémem číslování řeší situace, kdy odesilateli nedorazí potvrzovací paket a ten, domnívajíce se, že příjemce neobdržel datový paket, jej odešle znovu. Příjemce tak podle pořadového čísla pozná, že obdržel duplikát a může jej zahodit.

Taková komunikace by šla implementovat vlastními mechanismy. Není ale vhodné vytvářet řešení, jež lze zajistit technikami, které jsou již známy. Po krátké analýze požadavků se takovým řešením jeví široce podporovaný protokol TCP. Ten dokáže zajistit fázi přípravy spojení svým mechanismem *three-way handshake*, každý datový paket je potvrzován přijímací stranou a dále podporuje mechanismy pro řízení zahlcení a řeší ztrátu potvrzovacího paketu a následnou duplikaci dat.

Není tedy nutné v oblasti komunikace Senderu a Receiveru implementovat vlastní řešení, ale je možno použít existující protokol TCP.

Kapitola 5

Implementace

Vlastní implementace byla provedena v programovacím jazyce Python. K dnešnímu dni (duben 2012) jsou na stránkách Python komunity nabízeny ke stažení dvě základní stabilní verze interpretu, a to verze 2.7.x a 3.2.x. Překvapivě je starší verze nabízena přednostně a pro neznalé uživatele, kteří neví, kterou verzi zvolit, se na stránkách nachází rada, že mají zvolit verzi 2.7. Ačkoliv je Python 3 ve stabilní verzi již od února 2009, stále existuje mnoho modulů od vývojářů třetích stran, které nejsou s novou verzí kompatibilní. Pro některé se nová verze teprve připravuje, jiné se ji pravděpodobně nikdy nedočkají, protože již nejsou aktivně vyvíjeny. Z tohoto důvodu stránky doporučují starší verzi[10].

Na stejném místě můžeme najít také dokument popisující výhody a nevýhody obou verzí a doporučení, kterou verzi v jaké situaci použít. Ačkoliv jsou stále vydávány nové minoritní verze pro Python 2, opravují především bezpečnostní a jiné chyby. V aktivním vývoji je nyní Python 3. Obsahuje některé nové vlastnosti, které Python 2 postrádal. Jako příklad lze uvézt řetězce, se kterými se již ve výchozím stavu pracuje jako s řetězci ve znakové sadě Unicode. Dále se striktně rozlišuje mezi řetězci znaků a řetězci bytů, je přidána podpora pro anotace parametrů funkcí, zřetězování výjimek a další. Byly odstraněny problémové nebo nepoužívané standardní moduly, některé se sloučily a celkově došlo k jejich pročištění a zpřehlednění [22]. Pro programátory, kteří nejsou závislí na některém z modulů vývojářů třetích stran dostupného pouze pro verzi 2, je doporučeno použít nejnovější verzi. Protože se použití žádného takového modulu nepředpokládalo, bylo logické vyvíjet pro verzi 3.

Každý programovací jazyk má své výhody a nevýhody. Kromě Pythonu bylo ekvivalentně vhodné pro vývoj aplikace zvolit ještě jazyky C/C++ nebo Perl. Perl i Python jsou skriptovací jazyky a rozdíly mezi nimi jsou menší než rozdíly mezi Pythonem a C/C++. Ačkoliv nabízí Perl velké množství užitečných knihoven pro práci se síťovými službami, Python byl zvolen z důvodu lepší čitelnosti zdrojového kódu. Ze stejného důvodu byl Python upřednostněn i před C/C++. I pro programátory, kteří se teoreticky v budoucnu rozhodnou studovat kód aplikace, bude tato skutečnost nesmírnou výhodou. Celkově je vývoj v Pythonu oproti C/C++ rychlejší, více dynamický a flexibilní. Také přenositelnost, která je detailně rozebrána v samostatné kapitole 5.4, je na výborné úrovní.

Nevýhodou Pythonu je jeho menší výkon. Programy ve skriptovacích jazycích jsou zpravidla pomalejší než obdobné programy v kompilovaných jazycích. Protože se však nepočítalo, že bude aplikace extrémně výkonnostně náročná, nebral se tento problém jako příliš významný a výhody rychlosti vývoje, čitelné syntaxi a snadných budoucích úprav u Pythonu převážily. Jako zvláště výhodná se ukázala jeho flexibilta, kdy bylo nutné v průběhu implementace vytvořit několik ověřovacích prototypových verzí, jak je popsáno v kapitole 5.1.1.

5.1 Sender

Sender je dvouvláknová aplikace, která pro příjem a odesílání záznamů používá standardní síťové sokety a pro dočasné ukládání záznamů používá databázi. Samozřejmostí je podpora nejen IPv4, ale i IPv6.

V hlavním vlákně se provádí výběr záznamů z databáze, odesílání záznamů a odstraňování záznamů z paměti po úspěšném odeslání. Odesílání má na starost TCP klient. S podporou obou verzí protokolu IP pomáhá funkce `create_connection()`, která pracuje na vyšší úrovní než funkce `socket()`. Pokusí se ustavit spojení s Receiverem nejprve přes protokol IPv6, pokud neuspěje, použije IPv4.

Druhé vlákno je tvořeno výhradně UDP serverem, který přijímá záznamy z exportéru a ukládá je do databáze. Vytváření UDP serveru usnadnil modul *socketserver*. Pro podporu IPv4 i IPv6 bylo nutné z této třídy vytvořit dvě zděděné třídy, z nichž se v každé nastavila správná hodnota *address family*, tedy AF_INET a AF_INET6 (v pořadí). Při startu se aplikace nejprve pokouší vytvořit UDP server s IPv6 adresou, pokud neuspěje, přejde na IPv4.

Aplikace obsahuje několik nastavitelných parametrů. Jedná se o dvojici adresa – port, na které očekává záznamy přicházející z exportéru, dvojici adresa – port, na které by měl běžet TCP server Receiveru a kam se bezpečně odesílají záznamy. Dále možnost nastavení výpisu informací o běhu s různými úrovněmi závažnosti[1] a možnost nastavení délky čekání po neúspěšném připojení k Receiveru nebo po neúspěšném odeslání záznamu. Poslední možností je nastavení jména souboru databáze. Zde lze alternativně zadat speciální parametr `:memory:`, díky kterému se databáze nevytvoří fyzicky na disku, ale pouze jako objekt v operační paměti. Všechny tyto přepínače jsou podrobně popsány přímo v nápovědě aplikace.

Sender lze spustit bez parametrů a v takovém případě se použijí výchozí hodnoty. V reálném provozu je však nutné minimálně základní parametry zadat.

5.1.1 Systém dočasného ukládání záznamů

Vývoj v průběhu implementace

V průběhu implementace se musel několikrát zcela měnit systém ukládání záznamů.

Původním návrhem byla jednoduchá datová struktura typu fronta umístěná v operační paměti, kam se ukládaly záznamy postupně přicházející na vstupní rozhraní. Z této fronty se záznamy také četly při odebírání a následném odesílání přes výstupní rozhraní. Pouhá fronta v paměti však nestačila. Při výpadku linky odesílající data k Receiveru by se mohlo jednoduše stát, že by přicházející záznamy zahltily operační paměť, obzvláště v případech, ve kterých by záznamů bylo za sekundu velké množství, výpadek linky by trval delší dobu, paměť by byla malé velikosti nebo by došlo ke kombinaci těchto faktorů. Řešením se jevilo použití odkládacích souborů na disku. Záznamy z operační paměti by se na disk mohly ukládat po dosažení určité velikosti, popř. po vypršení maximálního časového limitu. Ve druhém případě bychom měli zaručené, že by se ztratily v případě pádu aplikace nebo celého operačního systému záznamy maximálně za posledních několik sekund, podle nastavené doby. Po podrobné analýze byl učiněn závěr, že použití tohoto systému dvojité fronty není vhodné z několika důvodů. Především by se v krajní situaci mohlo stát, že bychom ztratili obdržená data. Ztráta několik stovek až tisíců paketů vypadá na první pohled stěží tole-

[1] Jedná se o úrovně DEBUG, INFO, WARNING, ERROR a CRITICAL.

rovatelná. V celkovém dlouhodobém měřítku průměrně rozsáhlého NetFlow toku se však jedná o pouhý zlomek dat, který by byl za všech okolností pouze ve stínu následného výpadku způsobeného pádem aplikace nebo operačního systému. Této ztrátě se bohužel nedá zabránit žádným způsobem. Dalším problémem by byla komplikovanost souběžné práce se dvěma frontami a jejich vzájemná správná synchronizace.

Nasnadě tedy bylo najít vhodnější řešení. Z důvodu jednotného přístupu k frontě a požadavku na perzistenci dat, bylo nutné najít řešení, kde by se záznamy ukládaly přímo na disk. Další experimenty tak probíhaly se standardním Python modulem s názvem *shelve*. Podle dokumentace je *„shelf“* perzistentní objekt slovníkového typu uložený v souboru na disku. Každý záznam byl tedy ukládán do tohoto objektu a přistupovalo se k němu přes klíč, což bylo inkrementující se číslo typu integer. Nejvyšší číslo značilo konec fronty, nejnižší číslo začátek fronty. Nově příchozí záznam byl do souboru uložen jako záznam s klíčem s hodnotou o jedno vyšší než aktuálně nejvyšší číslo, tedy na konec fronty. Naopak při odebírání záznamů se odebíral záznam s nejnižším číslem. Při dlouhodobé činnosti aplikace se tak tento index posouval neustále do vyšších hodnot. Samo o sobě by tato vlastnost nebyla problém, modulu *shelve* však dělalo problém neustálé přidávání a odstraňování záznamů, při kterém rostla velikost souboru. Odstraněním záznamu s daným klíčem sice vedlo k znepřístupnění daného záznamu, ovšem ke zmenšení souboru o velikost záznamu nedošlo a další záznamy se neustále připisovaly sekvenčně na konec souboru. Jak se ukázalo, problém nebylo možné rozumným způsobem vyřešit a tento způsob ukládání záznamů se stal další slepou uličkou.

I přes obavu o degradaci výkonnosti celé aplikace se tak pokusy přesunuly k využití uložení záznamů do databáze. Python obsahoval až do verze 3 standardní modul rozhraní k databázi Berkeley. Tento modul byl však již ve verzi 2.6 označen jako zastaralý a s přechodem na verzi 3 došlo k jeho úplnému odstranění. Důvod k tomu kroku je popsán v dokumentu [4]. Ve zkratce se tento modul stal přítěží pro vývojáře, kteří mu museli věnovat neúměrný čas a úsilí. Nadále je vyvíjen jako externí modul.

Rovnocennou alternativou, která je i v aktuální verzi Pythonu přímo dostupná, je SQLite. Podle vývojařů se jedná o malou, rychlou a spolehlivou databázi, která je zároveň celosvětově nejrozšířenější co do počtu používaných kopií [25]. Další testy tedy byly prováděny s databází a protože se nakonec osvědčila jako vhodné řešení, je použita i ve finální verzi aplikace.

Zvyšování výkonnosti databáze

Při plném zatížení aplikace je i poměrně vysoká zátěž na databázi, a to především v podobě náročné souběžnosti operací čtení a zápisu, kdy obě operace jsou prováděny z různých vláken. SQLite nepodporuje uzamykání jednotlivých tabulek, čehož by aplikace nevyužila, protože pracuje pouze s jedinou tabulkou, a nepodporuje tedy ani uzamykání jednotlivých řádků tabulky, z čehož by naopak výkonnostně těžila. Souběžné čtení je možné v libovolném rozsahu. Pokud je však nutné provést operaci zápisu, SQLite zamyká celou databázi. Protože aplikace pracuje s databází podle neustále se opakujícího vzoru atomický zápis do databáze – atomické čtení z databáze, je v jednu chvíli prováděna vždy jen jedna operace. Každá z těchto atomických operací je dále zabalena do transakce, protože SQLite neumožňuje transakce zakázat a každá operace s databází musí být součástí nějaké transakce. Otevírání a uzavírání transakce je nezanedbatelně náročná operace, a proto nemožnost souběžného čtení a zápisu a nemožnost vypnutí transakcí dohromady degradují výkonnost databáze pro náš konkrétní účel.

Při hledání možností jak naopak výkon zvýšit se nabízelo upravit nastavení některých z vlastností databáze, tzv. PRAGMA [26]. Ve finální verzi aplikace jsou prováděny dvě změny oproti výchozímu nastavení, a to vypnutí *journal mode* a vypnutí *synchronous*. *Journal mode* nastavuje chování žurnálu. Při vypnutí se žádný žurnál nevytváří a není možné u transakcí používat rollback. Protože je však v aplikaci každá transakce sama o sobě atomickou operací, rollback je nepotřebná vlastnost a protože vytváření žurnálu s sebou nese větší nároky na systémové prostředky, především zvýšená frekvence diskových zápisů, je v aplikaci tato vlastnost vypnuta.

Největší dopad na výkon databáze však má vypnutí funkce *synchronous*. Ve výchozím nastavení je *synchronous* nastaveno na FULL a znamená to, že databáze čeká, až jsou všechna data opravdu zapsána na disk a teprve poté pokračuje. Toto nastavení je velmi bezpečné a zabraňuje, aby se databázový soubor poškodil (např. při pádu operačního systému nebo v případě výpadku napájení). Další možností je nastavení NORMAL, které je méně přísné než FULL, ale stále databáze u kritických zápisů čeká, až dojde k úplnému zapsání na disk. Poslední možností je vypnutí. V tomto režimu databáze pokračuje, jakmile data předá operačnímu systému. Nemůže tak dojít k poškození databázového souboru v případě pádu samotné aplikace, ale může dojít k poškození při pádu operačního systému nebo při výpadku napájení. Takové situace jsou však poměrně vzácné a protože se nejedná o databázi typu bankovního systému, kde by její poškození bylo naprosto nepřípustné, v našem případě lze takové chování tolerovat.

Při výchozím nastavení SQLite databáze (zapnuty funkce *synchronous* a *journal mode*) není aplikace schopna v reálném čase zpracovat více než cca 100 paketů za sekundu. Již při mírném překročení této hranice dochází k neustále narůstajícímu zpoždění odesílání. Záznamy jsou do databáze úspěšně ukládány, ale kvůli nutnosti čekání na dokončený zápis na disk a s ním spojené zamykání celé databáze nezbývá pro čtení záznamů výpočetní čas. Za jednotku času tak je více paketů přijato než odesláno. Při výrazném překročení pak aplikace již naprosto kolabuje a z příchozích záznamů je do databáze ukládána pouze část.

V případě, že jsou vypnuté funkce *synchronous* a *journal mode*, aplikace je schopná překročit hranici 1200 paketů za sekundu. Toto množství však ještě není stropem. Během testu při tomto počtu, bohužel, použitý generátor NetFlow paketů vytěžoval procesor na více než 70% a nezbývaly tak prostředky pro aplikace Senderu a Receiveru. Protože je však taková propustnost více než dostatečná pro reálné nasazení, další výkonnostní testy již neprobíhaly[2].

Tímto jednoduchým testem tak bylo potvrzeno, že vypnutí funkce *synchronous* má výrazný vliv na výkon databáze. V našem případě se jednalo přibližně o dvanáctinásobný nárust a podle [26] může znamenat až padesáti nebo i vícenásobný nárust výkonu.

Struktura databáze

Databázový soubor obsahuje pouze jedinou tabulku, kam se veškeré záznamy ukládají. Tato tabulka se skládá ze dvou sloupců. První z nich představuje primární klíč a jedná se o celé nezáporné číslo typu INTEGER. Druhý sloupec jsou samotná data záznamu, která se ukládají v binární podobě (v SQLite se jedná o typ BLOB).

Databáze představuje perzistentní frontu. Jak dochází k postupnému přidávání záznamů, zvětšuje se maximální primární klíč. Největší primární klíč tak představuje konec fronty. Naopak odebírání záznamů probíhá od nejnižších primárních klíčů a nejmenší z nich

[2]Na nejvytíženější sondě v univerzitní síti VUT se množství NetFlow záznamů ve špičce pohybuje kolem 550 za sekundu.

představuje začátek fronty. Primární klíč začíná u nově vytvořené databáze na indexu 1 a procesem přidávání a odebírání záznamů neustále narůstá do vyšších čísel. Ačkoliv se v průběhu vývoje předpokládalo, že bude nutné řešit dosažení maximálního čísla a jeho manuální snížení zpět na nulu z důvodu možného dosažení limitu databáze, opak je pravdou. Primární klíč typu INTEGER je v databázi uložen jako znaménkové 64 bitové číslo. Byla provedena následující úvaha: 64 bitové číslo představuje přibližně 1,8e19 čísel. Z celého rozsahu se využívá pouze polovina (kladná čísla s nulou). Tento rozsah je asi 9,2e18 čísel. Pokud by záznamy přicházely s frekvencí 1000 paketů za sekundu, za hodinu by to bylo 3600000 paketů a za rok 1314000000 paketů. Při vydělení rozsahu čísel možných uložit do databáze a počtu paketů, které by se uložily za rok při intenzitě 1000 paketů za sekundu, dojdeme k závěru, že dosažení maximální hodnoty by trvalo více než 7019309008 let. Dále pokud dojde k ukončení aplikace ve chvíli, kdy se v databázi nenachází žádný záznam, při jejím opětovném spuštění se začne počítat od primárního klíče s indexem 1. Dosažení limitu je tedy reálně nemožné.

Funkce databáze

Při spuštění aplikace se nejprve vyhledává existující databázový soubor. Pokud je nalezen, zjistí se, zda obsahuje záznamy, a pokud ano, aplikace si uloží minimální a maximální primární klíč (tedy začátek a konec fronty) a dále může s těmito indexy pracovat pro správné přidávání a odebírání záznamů.

Pokud databázový soubor neexistuje, vytvoří se společně s novou prázdnou tabulkou popsanou v kapitole 5.1.1.

Operace s databází vykonávají standardní SQL dotazy INSERT, SELECT a DELETE.

5.2 Receiver

Receiver je jednoduchá jednovláknová aplikace, která přijímá data od Senderu a provádí jejich přeposílání ke kolektoru. Přijímání zajišťuje TCP server, který nevyužívá modul `socketserver`, jak je tomu v případě UDP serveru u Senderu, ale je naprogramován pouze standardními operacemi soketů.

Receiver opět podporuje IPv6 i IPv4, a to jak při komunikaci se Senderem, tak při odesílání záznamů ke kolektoru.

Aplikace obsahuje několik nastavitelných parametrů podobných těm, použitých u Senderu. Jedná se o dvojici adresa – port, na které přijímá záznamy od Senderu, dvojici adresa – port, na které by měla běžet aplikace kolektoru a kam Receiver odesílá záznamy, a dále možnost nastavení výpisu informací o běhu s různými úrovněmi závažnosti[3]. Všechny tyto přepínače jsou podrobně popsány přímo v nápovědě aplikace.

Receiver lze spustit bez parametrů a v takovém případě se použijí výchozí hodnoty. V reálném provozu je však nutné minimálně základní parametry zadat.

5.3 Průběh komunikace

Data mezi Receiverem a Senderem se předávají pomocí spolehlivého protokolu TCP. Na vyšších vrstvách se nepoužívá žádný existující protokol. Bylo vytvořeno vlastní jednoduché řešení, protože nebylo nutné předávat složité struktury nebo metadata. Záznamy, které

[3] Jedná se o úrovně DEBUG, INFO, WARNING, ERROR a CRITICAL.

Sender příjme jako binární data, uloží v této podobě do databáze. Před odesláním jsou převedena do formátu Base64[4] a na konec se přidá znak LF (ASCII hodnota 10), díky čemuž Receiver pozná konec jednoho záznamu. Převedením do formátu Base64 dojde k nárustu velikosti dat o 33%, ale lze tímto způsobem bezpečně použít jako ukončovací znak ASCII hodnotu 10, protože je zaručeno, že v rámci samotných dat se takový symbol nemůže vyskytnout. Formát Base64 je pro předávání dat po síti poměrně oblíbený a lze se s ním často setkat v případech, kdy chce programátor zabránit, aby původně binární data byla chybně interpretována některým účastníkem komunikace jako řídicí nebo jiné speciální znaky.

Po úspěšném přijetí Receiverem, očekává Sender zprávu `OK` s ukončovacím znakem s ASCII hodnotou 10. Na přijetí potvrzení čeká pouze určitou dobu, poté označí zprávu za nedoručenou a pokusí se o její odeslání opět za určitou dobu. Tato doba je uživatelsky nastavitelná parametrem zadávaným při spuštění.

Receiver se dokáže vypořádat i se situací, kdy kvůli fragmentaci zprávy po cestě dorazí pouze určitá část. Těchto částí může být v podstatě libovolný počet, důležité je, aby se žádná z nich neztratila. To nám zabezpečuje protokol TCP. Po přijetí všech částí se sestaví původní zpráva a po převedení zpět do binární podoby se odešle kolektoru.

5.4 Přenositelnost

Python interpret standardně běží na operačních systémech Windows, Linux/Unix a Mac OS X. Vývoj aplikace probíhal na platformě Mac OS X. V Linuxu pak probíhalo testování v laboratorních podmínkách a v reálném nasazení. Provoz na těchto platformách je bezproblémový. Jelikož využívá aplikace pouze standardní moduly a není potřeba žádný software navíc. Pro spuštění stačí nainstalovat Python interpret verze 3.2.2 nebo vyšší. U nižších verzí Python 3 by aplikace měla fungovat také, nebyly však prováděny žádné explicitní testy.

Pod Windows aplikace v současné podobě nefunguje korektně. Při vývoji se nepočítalo s nutností běhu v tomto operačním systému, protože se v reálném nasazení tohoto typu používají většinou Unixové operační systémy. Pokud by však nastal požadavek na zpřístupnění této možnosti, nebylo by pravděpodobně nutné provádět dalekosáhlé změny. Jediným závažným nedostatkem, kvůli kterému aplikace nyní nefunguje, je použití funkce `inet_pton()`, která provádí převod IPv4 nebo IPv6 adresy ve formátu řetězce do binární podoby. Účelem je zjistit, zda adresa, kterou uživatel zadá při spuštění aplikace, je platná adresa IPv4 nebo IPv6. Pokud není, funkce vrací ve své implementaci v Pythonu výjimku. Windows tuto funkci podporuje od verze Vista (klientská verze) a Server 2008 (serverová verze), tedy přibližně od roku 2008 [17]. Podpora pro Python ovšem dodnes nebyla přidána. Tato chybějící vlastnost je evidována jako oficiální požadavek na vylepšení nadcházející verze. Podpora by měla být přidána ve verzi 3.3 [15].

Podobnou funkci jako `inet_pton()` obstarává také `inet_aton()`, která je v současné verzi Pythonu implementována i pro platformu Windows, avšak tato funkce nepodporuje IPv6.

[4] Způsob uložení binárních dat ve formě ASCII tisknutelných znaků.

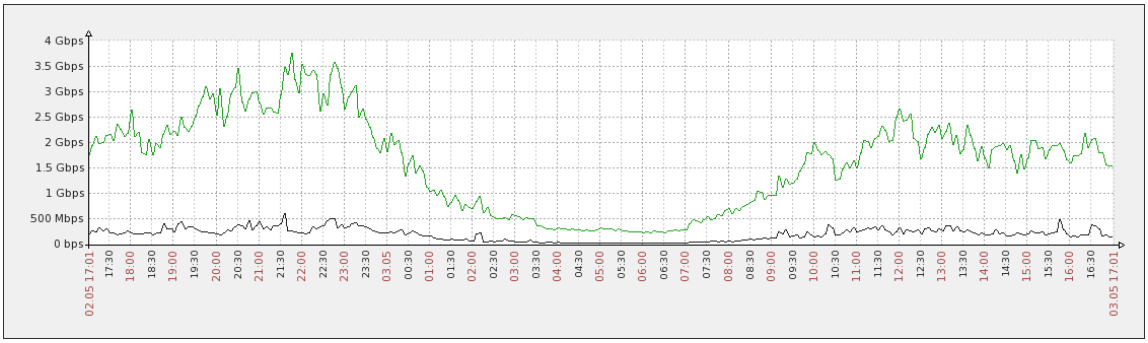
Kapitola 6

Průběh testování a výsledky testů

Testování probíhalo průběžně v rámci implementace na lokálním systému v prostředí operačního systému Mac OS X. Následné testování dokončené aplikace nebylo prováděno v laboratorním prostředí, ale přímo v produkčním prostředí univerzitní sítě VUT. Aplikace Senderu byla nasazena na sondu s operačním systémem Linux, distribuce CentOS verze 5. Aplikace Receiveru byla spuštěna na kolektoru s operačním systémem Linux, distribuce CentOS verze 6.

Testování probíhalo v průběhu jednoho týdne. Na následujících grafech jsou pro názornost ukázány výsledky během 24 hodin, a to ve dnech 2. května 2012 a 3. května 2012. Špičky se nacházejí především mezi sedmou hodinou večerní a půlnocí. Naopak největší útlum provozu nastává mezi druhou hodinou v noci a osmou hodinou ranní.

Na grafu 6.1 lze vidět rozložení množství dat, která tečou na monitorovaných linkách. V kombinaci IPv4 a IPv6 dosáhlo ve špičce toto množství 3,75 Gbps. Průměrné množství bylo 1,6 Gbps. V počtu paketů pak průměrně 208 710 paketů za sekundu a ve špičce 516 030 paketů za sekundu. Nativní IPv6 provoz z tohoto množství tvoří spíše zanedbatelnou část. Průměrně se jednalo o 180,36 Mbps, ve špičce 580,68 Mbps.

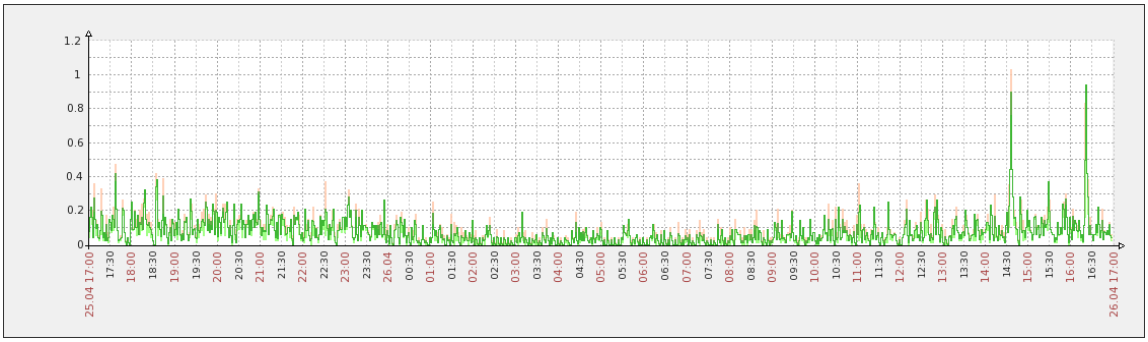


Obrázek 6.1: Množství IPv4 a IPv6 dat na monitorovaných linkách

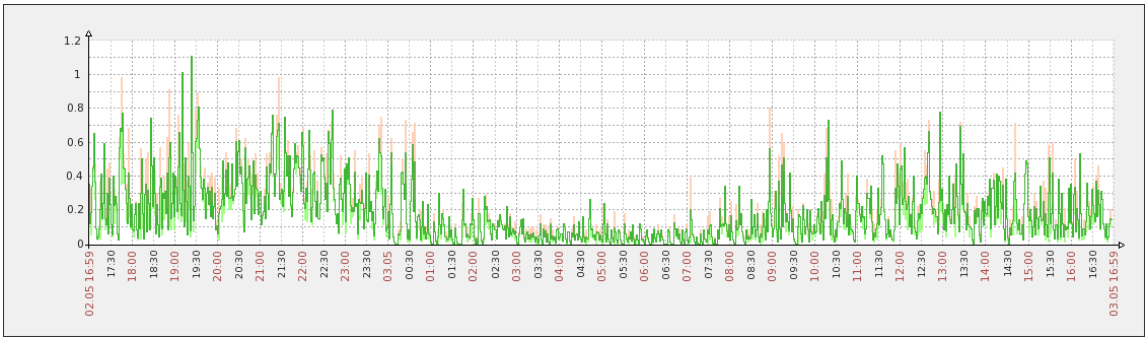
Velice důležitým údajem byl počet NetFlow záznamů, především jeho maximum. Ve špičce se pohybovalo kolem 550 flow za sekundu. Aplikace byla schopna při lokálním testování zpracovat více než 1200 paketů za sekundu a má tedy dostatečnou rezervu pro nasazení v ještě vytíženější síti.

Posledním sledovaným parametrem bylo vytížení procesoru sondy s použitím aplikace Senderu. Na grafu 6.2 lze vidět nejprve vytížení procesoru před použitím Senderu. Jedná

se o data sbíraná v průběhu dvaceti čtyř hodin 25. dubna 2012 a 26. dubna 2012. Toto vytížení není nikterak velké. Po nasazení Senderu však došlo k nárustu, jak je vidět na grafu 6.3. Mimo špičku je nárust zanedbatelný, naopak ve špičce poměrně značný. Tento efekt lze přičíst náročnosti zpracovávání paketů, kterých je ve špičce několikanásobně větší množství než v době útlumu provozu. Možné řešení tohoto problému je nastíněno v kapitole 7.



Obrázek 6.2: Vytížení procesoru sondy bez aplikace Senderu



Obrázek 6.3: Vytížení procesoru s použitím aplikace Senderu

Aplikace se osvědčila jako plně funkční a během testování nebyly nalezeny žádné problémy.

Kapitola 7

Možnosti rozšíření a optimalizace

Pro zamýšlenou funkčnost aplikace se ukázalo jako zbytečné zasahovat do samotných NetFlow záznamů. Celý záznam se transparentně, bez zásahů do samotných dat, přenese ze Senderu do Receiveru. Provádí se pouze převod do formátu Base64 a následný převod zpět. Protože však lze do tohoto formátu převést jakákoliv binární data, která Sender obdrží, je možné stejně jako NetFlow záznamy přenést bezpečně i data jiného původu. Takové využití lze uplatnit obecně všude, kde je nutné spolehlivě přenést UDP pakety, tedy určitá simulace spolehlivého transportu u TCP. Během implementace tedy vznikla rozšířená funkcionalita, která nebyla původně zamýšlena.

NetFlow záznamy jsou citlivé informace a možnost jejich odposlechu jistě není pro administrátora vítaná skutečnost. Velmi užitečným rozšířením do budoucna by tedy mohlo být šifrování přenášených dat pomocí některého z kryptografických protokolů (SSL, TLS a jiné).

Receiver by pro zvýšení bezpečnosti měl být také schopen provádět autentizaci Senderu. Potenciální útočník by bez této vlastnosti mohl jednoduše posílat náhodně generované nebo naopak záměrně modifikované NetFlow záznamy, které by se ukládaly do kolektoru, a docházelo by k ovlivňování výsledků statistik provozu. Autentizace by tomuto problému zabránila.

V případě, že by nestačil výkon současné aplikace nebo by bylo nutné snížit nároky na vytížení procesoru, bylo by možné přepsat části kódu do kompilovaného jazyka, především do C nebo C++. Musela by se provést výkonnostní analýza a identifikace částí kódu spotřebovávající nejvíce systémových prostředků. Přepis pouze části kódu je rozumným kompromisem mezi současným stavem a aplikací kompletně přepsanou v kompilovaném jazyku.

Kapitola 8

Závěr

Bakalářská práce se zabývala monitorováním datových sítí na bázi IP flow. Především byla popsána architektura monitorování na bázi NetFlow. Byly vysvětleny základní pojmy, způsoby zapojení monitorovacích sond a podrobně popsán protokol NetFlow verze 9. Práce se zmiňuje i o alternativách protokolu NetFlow.

V další části byla zhodnocena slabá místa monitorovacích systémů a bylo navrženo konceptuální řešení. Toto konceptuální řešení bylo implementováno a implementace detailně popsána.

V rámci bakalářské práce vznikla aplikace, která umožňuje transparentní přenos NetFlow záznamů ze sondy do kolektoru. I v případě výpadku linky do kolektoru jsou data bezpečně uchovávána a přeposlána po opětovném zprovoznění linky. Administrátoři tak mohou zpětně nahlížet do monitorovacích dat i ve chvílích, kdy došlo v síti k poruše a kdy bez této aplikace by neměly přístup k datům právě z důvodu výpadku. Aplikace je vytvořena v programovacím jazyce Python, využívá databázi SQLite a pracuje univerzálně, tzn. dokáže transparentně přeposílat nejen NetFlow záznamy, ale i libovolná jiná data.

Testování probíhalo přímo v reálném nasazení v rámci univerzitní sítě VUT. Data byla sbírána na sondě, která je v této síti nejvytíženější a počet NetFlow záznamů se ve špičce pohybuje okolo 550 za sekundu. Aplikace se osvědčila jako plně funkční a během testování nebyly nalezeny žádné problémy.

Na práci je možno v budoucnu navázat rozšířením funkčnosti aplikace a její optimalizací. Především se jedná o implementaci autentizace a šifrování přenášených dat. NetFlow záznamy jsou poměrně citlivá data, která by se dala snadno zneužít, a možnost jejich odposlechu jistě není pro administrátora vítaná skutečnost. Dále je možná optimalizace s cílem snížit nároky na výpočetní výkon.

Literatura

[1] Astashonok, S. *FProbe* [online]. 2002, Last Update 2011-05-03 [cit. 2011-12-05]. Dostupné na: <http://sourceforge.net/projects/fprobe/>.

[2] Brownlee, N., Mills, C. a Ruth, G. *Traffic Flow Measurement: Architecture.* October 1999. RFC, 2722. Dostupné na: <http://tools.ietf.org/html/rfc2722>.

[3] Caligare. *Caligare Flow Inspector: NetFlow monitor and analyzer* [online]. ©2003-2011 [cit. 2012-04-15]. Dostupné na: <http://www.caligare.com/netflow/caligare_flow_inspector.php>.

[4] Cannon, B. *Standard Library Reorganization* [online]. 01-Jan-2007, Post-History: 28-Apr-2008 [cit. 2012-04-02]. Dostupné na: <http://www.python.org/dev/peps/pep-3108/>.

[5] Cisco Systems. *Cisco IOS NetFlow* [online]. 2011 [cit. 2011-12-12]. Dostupné na: <www.cisco.com/web/go/netflow>.

[6] Citrix Systems. *AppFlow Specification* [online]. Updated May 19 2011 [cit. 2012-04-15]. Dostupné na: <http://www.appflow.org/appflow-spec-v1>.

[7] Claise, e. *Cisco Systems NetFlow Services Export Version 9.* October 2004. RFC, 3954. Dostupné na: <http://www.ietf.org/rfc/rfc3954.txt>.

[8] Datacom Systems Inc.. *TAP vs. SPAN* [online]. ©1992-2009 [cit. 2011-12-12]. Dostupné na: <http://www.datacomsystems.com/partners/auth/content/Tap-Vs-Span.pdf>.

[9] Fluke Corporation. *OptiView(R) NetFlow Tracker* [online]. ©2006-2012 [cit. 2012-04-15]. Dostupné na: <http://www.flukenetworks.com/enterprise-network/network-monitoring/OptiView-NetFlow-Tracker>.

[10] Foundation, P. S. *Download Python* [online]. ©1990-2011 [cit. 2012-04-02]. Dostupné na: <http://www.python.org/getit/>.

[11] Haag, P. *NFDUMP: Netflow processing tools* [online]. 2004, Last Update 2012-03-13 [cit. 2012-04-15]. Dostupné na: <http://sourceforge.net/projects/nfdump/>.

[12] Huawei Technologies. *Technical White Paper for NetStream* [online]. ©2007 [cit. 2012-04-15]. Dostupné na: <http://www.huawei.com/products/datacomm/pdf/view.do?f=65>.

[13] Internet Engineering Task Force. *IP Flow Information Export (ipfix)* [online]. 2012-03-06 [cit. 2012-04-15]. Dostupné na: <http://datatracker.ietf.org/wg/ipfix/>.

[14] INVEA-TECH a.s.. *FlowMon* [online]. ©2007-2011 [cit. 2011-12-12]. Dostupné na: <http://www.invea.cz/produkty-sluzby/flowmon>.

[15] jason.coombs. *Add inet_ntop and inet_pton support for Windows* [online]. 2009-10-19 23:20, last changed 2011-09-01 18:29 [cit. 2012-04-02]. Dostupné na: <http://bugs.python.org/issue7171>.

[16] Juniper Networks. *Juniper Flow Monitoring* [online]. ©2011 [cit. 2012-04-15]. Dostupné na: <http://www.juniper.net/us/en/local/pdf/app-notes/3500204-en.pdf>.

[17] Microsoft. *InetNtop function* [online]. Build date: 3/6/2012 [cit. 2012-04-02]. Dostupné na: <http://msdn.microsoft.com/en-us/library/cc805843%28VS.85%29.aspx>.

[18] NetOptics(R). *Comparing the use of Taps and Span Ports* [online]. 2007 [cit. 2011-12-12]. Dostupné na: <http://www.netoptics.com/sites/default/files/support/taps-and-span-ports.pdf>.

[19] NetUP Inc.. *NDSAD* [online]. 2004, Last Update 2009-07-17 [cit. 2011-12-05]. Dostupné na: <http://sourceforge.net/projects/ndsad/>.

[20] ntop. *NProbe*^{TM} *v6* [online]. ©1998-2011 [cit. 2011-12-12]. Dostupné na: <http://www.ntop.org/products/nprobe/>.

[21] Phaal, P. a Lavine, M. *SFlow Version 5* [online]. July 2004 [cit. 2012-04-15]. Dostupné na: <http://www.sflow.org/sflow_version_5.txt>.

[22] Piercy, S. *Should I use Python 2 or Python 3 for my development activity?* [online]. Poslední editace 2012-04-11 02:52:11 [cit. 2012-04-15]. Dostupné na: <http://wiki.python.org/moin/Python2orPython3>.

[23] Quittek, J. et al. *Requirements for IP Flow Information Export (IPFIX).* October 2004. RFC, 3917. Dostupné na: <http://www.ietf.org/rfc/rfc3917.txt>.

[24] Rajahalme, J. et al. *IPv6 Flow Label Specification.* March 2004. RFC, 3697. Dostupné na: <http://www.ietf.org/rfc/rfc3697.txt>.

[25] SQLite. *Most Widely Deployed SQL Database* [online]. [2007?] [cit. 2012-04-02]. Dostupné na: <http://sqlite.org/mostdeployed.html>.

[26] SQLite. *PRAGMA Statements* [online]. [2011?] [cit. 2012-04-02]. Dostupné na: <http://www.sqlite.org/pragma.html>.

Příloha A

Obsah CD

- Text této bakalářské práce ve formátu PDF, včetně zdrojových kódů pro LaTeX
- Skript aplikace Senderu pro Python verze 3
- Skript aplikace Receiveru pro Python verze 3
- Upravená verze skriptu aplikace Receiveru pro Python verze 2.6
- Python interpret verze 3.2.2 pro Windows
- Python interpret verze 3.2.2 pro Mac OS X 10.3 až 10.6
- Python interpret verze 3.2.2 pro Mac OS X 10.6 a 10.7
- Zdrojové kódy Python interpretu verze 3.2.2