MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Proudové zpracování dat: Bezpečnostní monitorování sítě pomocí Apache Samza

DIPLOMOVÁ PRÁCE

Bc. Martin Laštovička

Brno, jaro 2016

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Tomáš Jirsík

Poděkování

Rád bych poděkoval RNDr. Tomáši Jirsíkovi za odborné vedení, konzultace a dobré rady při psaní této práce. Poděkování patří také celému týmu CSIRT-MU, který mi poskytl možnost podílet se na výzkumném projektu a vytvořil skvělé zázemí pro psaní této práce. Zvláštní poděkování si zaslouží také moje rodina za podporu během celé doby mého studia.

Shrnutí

Tato práce se zabývá využitím vysoce škálovatelného paradigmatu proudového zpracování dat pro bezpečnostní monitorování síťového provozu a jeho výhodami vůči dávkovému zpracování. V praktické části práce je provedeno výkonnostní testování nejpoužívanějších frameworků pro proudové zpracování dat a následně je v Apache Samza implementována metoda detekce útoků na autentizaci vzdálené plochy, jejíž výsledky jsou poté srovnány s komerčním produktem Flowmon ADS.

Klíčová slova

NetFlow, Bezpečnostní monitorování sítě, Proudové zpracování dat, Apache Samza, Detekce útoků, RDP

Obsah

1 **Úvod** . . . 3
2 **Bezpečnostní monitorování sítě** . . . 5
2.1 *NetFlow* . . . 6
2.1.1 Životní cyklus síťového toku . . . 6
2.1.2 Architerktura monitorování sítě pomocí NetFlow . . . 7
2.2 *Zpracování síťových toků a detekční metody* . . . 8
2.2.1 NfSen . . . 9
2.2.2 Flowmon ADS . . . 11
3 **Proudové zpracování dat** . . . 14
3.1 *Odlišnosti proudového paradigmatu* . . . 14
3.1.1 Posuvná okna . . . 15
3.1.2 Vzorkování a dávkování . . . 15
3.2 *Zdroje dat* . . . 16
3.2.1 Apache Kafka . . . 16
4 **Systémy pro zpracování dat v reálném čase** . . . 18
4.1 *Apache Samza* . . . 18
4.1.1 Správa clusteru . . . 18
4.1.2 Datový a výpočetní model . . . 20
4.2 *Apache Spark* . . . 21
4.3 *Apache Storm* . . . 22
5 **Výkonnostní testování vybraných frameworků** . . . 24
5.1 *Testovací scénáře* . . . 24
5.2 *Architektura výkonnostního testování* . . . 25
5.2.1 Testovací data . . . 25
5.2.2 Hardware a propojení strojů . . . 26
5.2.3 Architektura testů . . . 27
5.3 *Implementace testovací metod v Apache Samza* . . . 29
5.4 *Testování frameworku Apache Samza* . . . 31
5.4.1 Výsledky testů . . . 31
5.5 *Porovnání s Apache Spark a Apache Storm* . . . 34
6 **Detekce útoků na autentizaci RDP** . . . 36
6.1 *Časové okno* . . . 37

6.2 *Detekce útoku* 38
6.3 *Testování a porovnání s Flowmon ADS* 38
7 **Závěr** 42
Literatura 46
Obsah přiloženého CD 47

Kapitola 1

Úvod

Neustále se zvětšující objem síťové komunikace je neoddiskutovatelným jevem provázející rozvoj moderních technologií. Z pohledu uživatele se nejedná o nic špatného, naopak větší množství služeb v lepší kvalitě je vítaným přínosem. Z pohledu bezpečnostní analýzy je ovšem situace značně odlišná, protože velký objem dat klade obrovské nároky na výpočetní výkon.

Analýza všech paketů procházejících sítí je výpočetně náročná a s postupným nárůstem podílu šifrované komunikace v síťovém provozu přestává být prakticky použitelná [24]. Tyto nedostatky řeší monitorování provozu pomocí síťových toků (flow), které se zaměřuje pouze na hlavičky paketů a ne jejich datový obsah. Tok je z definice sdružení paketů obsahujících stejnou zdrojovou/cílovou IP adresu, zdrojový/cílový port a použitý protokol. Tyto informace jsou doplněny o časová razítka začátku a konce toku, počtu přenesených paketů a bajtů, případně další položky [11]. Tím se objem dat, která jsou podrobována analýze rapidně snižuje, což umožňuje nasazení síťového monitorování i do prostředí páteřních sítí s velkým datovým průtokem.

Technologie síťových toků je velmi oblíbená a rozšířená. Ovšem i u současných nástrojů pro bezpečnostní analýzu pomocí síťových toků nalezneme nedostatky. Tím hlavním je dávkový mechanismus zpracování. Metody založené na NetFlow [26], jako jsou například otevřený systém NfSen [20] nebo na univerzitě využívaný systém Flowmon ADS [16] od společnosti Flowmon Networks pracují na principu sběru síťových dat, která jsou následně v pravidelných časových intervalech (typicky 5 minut) podrobována analýze. To ovšem způsobuje časové prodlevy mezi útokem, jeho detekcí a následnou reakcí.

Řešení takto zpožděné detekce představuje proudové zpracování dat, které umožňuje zkontrolovat každý síťový tok v okamžiku jeho vzniku a odhalení útoků tak probíhá v reálném čase. Nasazení systémů pro proudové zpracování dat pro bezpečnostní analýzu v současnosti není příliš rozšířené a cílem mé diplomové práce je prozkoumat možnosti uplatnění

takového systému v praxi.

Teoretická část práce se nejprve věnuje paradigmatu proudového zpracování dat v reálném čase a porovnává ho se zpracováním dávkovým. Následně jsou analyzovány tři nejpoužívanější frameworky proudového zpracování z hlediska potřeb bezpečnostní analýzy. Ta klade na zpracující systémy velmi specifické požadavky z hlediska výkonu a typu příchozích dat. Systémy Apache Samza [3], Apache Storm [1] a Apache Spark [28] jsou proto porovnány vůči těmto požadavkům a následuje jejich výkonnostní porovnání.

Praktickou částí mé práce je zapojení systému Apache Samza do výkonnostního srovnání frameworků pro proudové zpracování dat. Dalším krokem práce je implementace metody detekce síťového útoku pomocí Apache Samza a porovnání výsledků jejích detekcí se stávajícím dávkovým systémem.

Kapitola 2

Bezpečnostní monitorování sítě

V této kapitole si představíme současné nástroje používané v oblasti monitorování sítě a bezpečnostních analýz. Popis těchto metod se zaměří především na oblasti, ve kterých může představovat využití proudového paradigmatu posun vpřed.

Monitorování síťového provozu je v podstatě monitorování paketů. Podle zpracovávaných údajů z jednotlivých paketů se vyvinuly tři základní typy analýzy. Jde o hloubkové a mělké inspekce paketů a analýza síťových toků [24].

Během hloubkové inspekce paketů je zpracován paket jako celek. Touto metodou je tedy možné zpracovat datovou část paketu a podrobit ji vyhledávání podezřelých vzorů. To s sebou přináší vysoké požadavky na výpočetní výkon a v sítích s velkým objemem přenesených dat přestává být prakticky použitelné. V současné době je navíc trendem čím dál více komunikace šifrovat. To činí hloubkovou analýzu nemožnou, protože obsah paketu je vypadá jako série náhodných čísel a porovnávat ho se známými vzory pak nedává smysl.

Oba zmíněné problémy řeší mělká inspekce paketu. Její zájmovou oblastí je pouze hlavička paketu obsahující pouze směrovací informace, číslo protokolu a kontrolní součet. Zpracovávaných dat je tak podstatně méně a navíc mají pevný formát umožňující efektivní procházení. Nevýhodou mělké inspekce je chybějící kontext. Pakety jsou analyzovány jednotlivě a rozhodnutí o případné akci tak musí být provedeno na základě předem definovaných pravidel závisejících pouze na informacích obsažených v každém paketu. Využití nachází především při filtrování komunikace, kdy směrovač rozhoduje, do kterého portu paket odešle. Na základě blacklistu IP adres tak může například zahazovat veškerou komunikaci útočníka, nebo omezit přístup ke stroji pouze pro daný rozsah adres.

Poslední variantou zpracování paketů je tvorba síťových toků. Pakety tak nejsou zpracovávány jednotlivě, ale jsou nejprve agregovány podle zdrojové a cílové IP adresy, zdrojového a cílového portu a použitého protokolu. Tím se jednotlivé pakety dostávají do širších souvislostí, což

umožňuje provádět složitější analýzy a detekce podezřelé komunikace. Celý systém vzniku toku a jeho zpracování si podrobně představíme na v současnosti nejpožívanějším protokolu síťových toků, NetFlow, a jeho novější variantě IPFIX.

2.1 NetFlow

Historické kořeny protokolu NetFlow sahají do roku 1996, kdy v Cisco Systems vytvořili nový způsob rychlého směrování paketů pomocí síťových toků. Vyhledávání ve směrovací tabulce routeru tak bylo potřeba pouze pro první paket toku a ostatní pakety se řídily podle směrování příslušného toku.

2.1.1 Životní cyklus síťového toku

Síťový tok je v NetFlow definován jako n-tice (zdrojová IP adresa, cílová IP adresa, zdrojový port, cílový port, L3 protokol, třída služby) [10]. Pro účely směrování byla v původní verzi přímo do toku zahrnuta ještě informace o cílovém rozhraní směrovače. Tyto údaje jsou povinné, ale tok obvykle obsahuje mnohem více informací v závislosti na použité verzi NetFlow.

Životní cyklus každého toku je definován příchodem prvního paketu s unikátní kombinací výše uvedených položek. Tak je vytvořen nový tok a každý další příchozí paket se shodnou n-ticí je přiřazen tomuto toku. Uzavření toku je definováno splněním jedné z následujících podmínek; buď je detekován paket náležející toku, který obsahuje příznak ukončení komunikace (například TCP příznaky FIN finish a RST reset), nebo v průběhu předem stanoveného časového intervalu (inactive flow timer) nepřišel žádný paket daného toku (obvyklé nastavení je 15 sekund), a konečně v případě dlouho trvajících toků je tento tok ukončen po překročení maximální možné délky toku (active flow timer – defaultně v Cisco směrovačích nastaveno na 30 minut). Po uzavření toku jsou spolu s tokem uloženy doplňující informace – časová známka začátku toku, doba trvání, počet přenesených paketů, bajtů, čísla autonomních systémů zdroje i cíle komunikace, TCP příznaky a další v závislosti na verzi NetFlow.

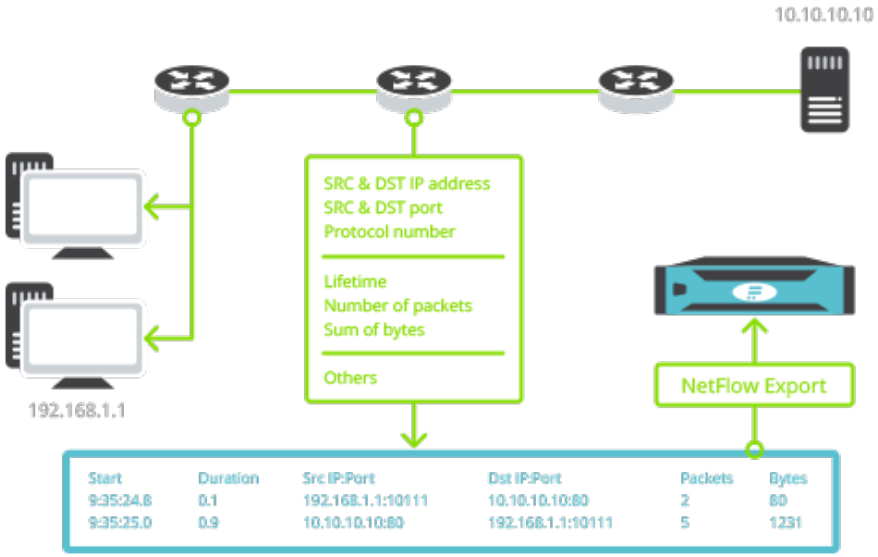
Z hlediska bezpečnostní analýzy a obzvláště zpracování v reálném čase je ukončení toku velice důležité. Celkové zpoždění od příchodu prvního paketu do uzavření toku a jeho export k analýze se rovná celé délce toku plus čekání na uzavření. Tato doba však nemůže překročit interval stanovený pro dělení paketů:

$$delay = flow\_duration + inactive\_timeout \leq active\_timeout$$

Příkladem útoku, jehož detekce může být touto prodlevou zpožděna, je distribuované zahlcení serveru obrovským množstvím požadavků s uměle sníženou propustností. Při správném nastavení odezvy útoku se neuzavře ani síťový tok ani TCP spojení se serverem, který tato spojení musí udržovat. Tím lze teoreticky dosáhnout pomocí flow těžko detekovatelného DoS útoku.

2.1.2 Architerktura monitorování sítě pomocí NetFlow

Tvorba síťových toků probíhá na síťové sondě. Jedná se o zařízení napojené přímo na fyzickou linku schopné zachytávat přenášené pakety a podle jejich hlaviček vytvářet toky. Dříve byla tato funkcionalita implementována přímo ve směrovačích, ale vzhledem k výpočetní náročnosti současného směrování a počítání toků, se sondy vyčlenily do samostatného síťového prvku.



Obrázek 2.1: Schéma monitorování sítě pasivní sondou a export dat na kolektor [17]

Výhodou fyzické sondy je také možnost monitorovat síťový provoz pasivně. Sonda je tak napojena na linku, ale do provozu nijak nezasahuje a pro ostatní zařízení je prakticky neviditelná.

Jiná možnost řešení výpočetního zatížení směrovače, která se kdysi používala, bylo vzorkování provozu. Do toků tak byl zpracováván pouze

každý n-tý paket, ovšem toto řešení bylo z hlediska bezpečnostního monitorování velmi nevhodné, jelikož narušovalo činnost detekčních metod.

Poté, co je tok vytvořený, je nutné ho odeslat do komponenty zvané kolektor ke zpracování. Odesílání toků má na starosti exportér, který je součástí sondy a někdy se tyto dva pojmy ztotožňují. K exportování toku zpravidla nedochází okamžitě po jeho vytvoření, ale nejprve je vytvořen balík 30-50 toků, které jsou přes protokol UDP odeslány kolektoru. Taková agregace je důležitá z hlediska výkonu, protože na monitorovaných linkách běžně vznikají tisíce toků za vteřinu a jejich samostatné odesílání by bylo neefektivní. Schéma znázorňující celý proces tvorby síťového toku a zapojení jednotlivých síťových prvků je znázorněn na obrázku 2.1.

Hlavním úkolem kolektoru je příchozí flow data uložit a zpřístupnit dalším aplikacím k analýze. Ukládání dat probíhá dávkově, kdy kolektor přijímá všechna příchozí data a v pravidelných časových intervalech (obvykle 5 minut) je uloží do souboru. Samotné zpracování dat a jejich analýzu je tak možné provádět pouze nad daty již uloženými na disku. V současné době je kolektor typicky samostatné fyzické zařízení, které je dedikované pouze na příjem a zpracování dat.

2.2 Zpracování síťových toků a detekční metody

Analýza flow dat je závislá na informacích obsažených v každém toku a ty jsou dále závislé na dané verzi Netflow. Protokol Netflow existuje v 9 různých variantách, z nichž nejpoužívanější jsou v5 (historicky nejrozšířenější) s pevnou strukturou dat a poslední verze v9 umožňující dynamické rozšíření obsažených položek. Informace, které obsahuje každý tok si ukážeme na struktuře Netflow v5 [9] v tabulce 2.1.

Tyto údaje jsou základem všech dalších verzí a obvykle je přebírají i verze 9 a nástupce Netflow, protokol IPFIX. Struktura Netflow v9 je určená šablonou [8] obsahující celkový počet položek toku a pro každou položku její typ a délku v bajtech. Je tak možné vytvořit vlastní strukturu obsahující více informací, které jsou schopné sondy získat, nebo naopak vypustit pro danou službu nezajímavé informace a snížit tak výpočetní nároky na jednotlivé komponenty.

Nástupcem Netflow je protokol IPFIX [11], který rozvíjí rozšiřitelný formát šablon a nabízí naprostou volnost ohledně zpracovávaných položek včetně informací z vyšších síťových vrstev. Pro bezpečnostní analýzy jsou velmi zajímavé například údaje z HTTP protokolu (hostname a URL) nebo obsah DNS dotazů, které je možné do IPFIX toku uložit.

Zkratka	Popis	Počet bajtů
srcaddr	Zdrojová IP adresa	4
dstaddr	Cílová IP adresa	4
nexthop	IP adresa následujícího směrovače	4
input	SNMP index vstupního rozhraní	2
output	SNMP index výstupního rozhraní	2
dPkts	Počet paketů v toku	4
dOctets	Celkový počet bajtů přenesených v toku	4
First	Timestamp příchodu prvního paketu toku	4
Last	Timestamp příchodu posledního paketu toku	4
srcport	Číslo zdrojového TCP/UDP portu	2
dstport	Číslo cílového TCP/UDP portu	2
tcp_flags	Logický součet všech TCP příznaků	1
prot	Číslo IP protocolu	1
tos	Typ IP služby (ToS)	1
src_as	Číslo autonomního systému zdroje	2
dst_as	Číslo autonomního systému cíle	2
src_mask	Maska zdrojové adresy	1
dst_mask	Maska cílové adresy	1

Tabulka 2.1: Obsah síťového toku protokolu Netflow v5

2.2.1 NfSen

V současnosti nejrozšířenějším open source nástrojem pro analýzu síťových toků je balík NFDUMP [19] a jeho grafická nadstavba s uživatelským rozhraním NfSen [20]. Tento balík obsahuje nástroje pro kompletní zpracování síťového provozu od jeho zachytávání a ukládání flow dat (netflow capture daemon) až po jejich analýzu (nástroj nfdump). Celý balík NFDUMP je navržen pro práci s daty ve formátu Netflow v5, v7 a v9, a tudíž jej nelze použít pro analýzu dat vyšších síťových vrstev obsažených v IPFIX.

Jádrem práce s flow daty je tedy nástroj nfdump, jehož možnosti si podrobně představíme na jeho jednotlivých funkcích:

- **Projekce** – nfdump umožňuje nakonfigurovat výstupní formát a omezit tak množství zobrazovaných informací pouze na vybranou podmnožinu položek Netflow. Kromě této základní projekce lze navíc zobrazit tři základní statistiky o rychlosti přenosu dat v daném toku (počet bitů za vteřinu , bajtů za vteřinu a průměrný počet bajtů v paketu).

- **Agregace** – toky je možné agregovat podle libovolné kombinace položek zdrojové/cílové IP adresy a zdrojového/cílového portu. V případě IP adresy se nemusí jednat o konkrétní adresu, ale lze nastavit i adresu podsítě specifikovanou pomocí CIDR notace *ip_sítě/délka_masky*.

- **Filtr** – výběr toků splňujících dané podmínky je realizován nad libovolnou položkou toku včetně statistik popsaných v projekci.

- **TOP N statistiky** – výběr n agregovaných toků, které jsou seřazeny podle počtu toků, paketů, bajtů, paketů za vteřinu, bajtů za vteřinu nebo bajtů na paket.

- **Anonymizace toků** – kryptografická anonymizace IP adres pomocí Crypto-PAn algoritmu [14], který adresy mapuje jedna k jedné, přičemž zachovává shodu prefixu adres ze stejné sítě.

I když je možné všechny výše zmíněné nástroje používat samostatně, pro praktické využití je vhodné mít k dispozici ucelený framework s grafickým rozhraním. Takovým systémem je Netflow Sensor (NfSen) [20], který pracuje nad již uloženými síťovými toky a umožňuje uživateli získat přehled o dění v síti pomocí grafů a statistik provozu a především provádět vlastní analýzy provozu.

NfSen poskytuje tři druhy přístupu k bezpečnostní analýze:

- Manuální analýza – NfSen nabízí grafické rozhraní nad NfDump, ve kterém je možné využívat veškerou funkcionalitu popsanou výše bez nutnosti znalosti syntaxe argumentů NfDump. Výběr zdrojových dat i jejich časové období totiž probíhá klikáním do obrazové reprezentace a výběr typu analýzy je rovněž prováděn pomocí dobře popsaných textových polí a drop-down listů. Díky tomu se může analytik zaměřit přímo na samotný filtr komunikace nemusí řešit zadávání argumentů do příkazové řádky.

- Alert – tato varianta představuje plně automatické zpracování flow dat. Funguje na principu počítání statistik provozu a jejich porovnání s předem nastaveným prahem. Dané statistiky je možné počítat pro jednotlivé položky síťového toku (IP adresy, porty, protokol, autonomní systém, atd.) i pro celkový objem komunikace. Pokud některá ze statistik překročí nastavený práh, je vykonána akce na základě spuštění tzv. triggeru. Ten přesně určí kdy na konkrétní překročení reagovat a jaká bude výsledná akce, která může být odeslání emailu nebo zavolání pluginu NfSenu.

- Plugin NfSenu – poslední varianta bezpečnostní analýzy poskytuje rozhraní pro automatické spouštění libovolného procesu zpracování dat. Plugin je program napsaný v jazyce Perl, který je integrován do NfSenu a sestává se z backendu zodpovědného za zpracování dat a frontendové PHP stránky pro prezentaci dat a ovládání pluginu. Plugin je spuštěň příchodem nových dat, spuštěním alertu nebo akcí uživatele ve webovém rozhraní. Detailní popis pluginů a jejich implementace je popsán v mé bakalářské práci [22]. Pro účely bezpečnostní analýzy je důležitá volnost při zpracování příchozích flow dat, která je omezena pouze výkonem stroje (balík dat musí být zpracován dříve než přijde další, v prostředí NfSenu je tento interval typicky 5 minut).

Nástroje z rodiny NfDump a NfSen tedy umožňují sběr Netflow dat, jejich uložení a následné zpracování. Ovšem schopnost odhalovat kyberútoky je závislá pouze na znalostech a zkušenostech analytika provozujícího tento systém.

2.2.2 Flowmon ADS

V této části si představíme reprezentanta komerční sféry v oblasti bezpečnostního monitorování sítě, produkt Flowmon ADS [16] (Anomaly Detection System) od společnosti Flowmon Networks, a.s., který je v současnosti využíván i na půdě Masarykovy Univerzity.

Tento produkt není založen čistě na Netflow, ale k detekcím využívá také informace z protokolů IPFIX, jFlow a NetStream, díky čemuž může odhalit více útoků pomocí informací z vyšších síťových vrstev.

Ve verzi ADS 7.02.00 [15] je implementováno 40 detekčních metod rozdělených do 5 skupin. Zaměřím se tedy na popis společných vlastností metod jednotlivých skupin a následně na detekci útoků proti autentizaci protokolu RDP (Remote Desktop Protocol), kterou budu dále zpracovávat v praktické části práce.

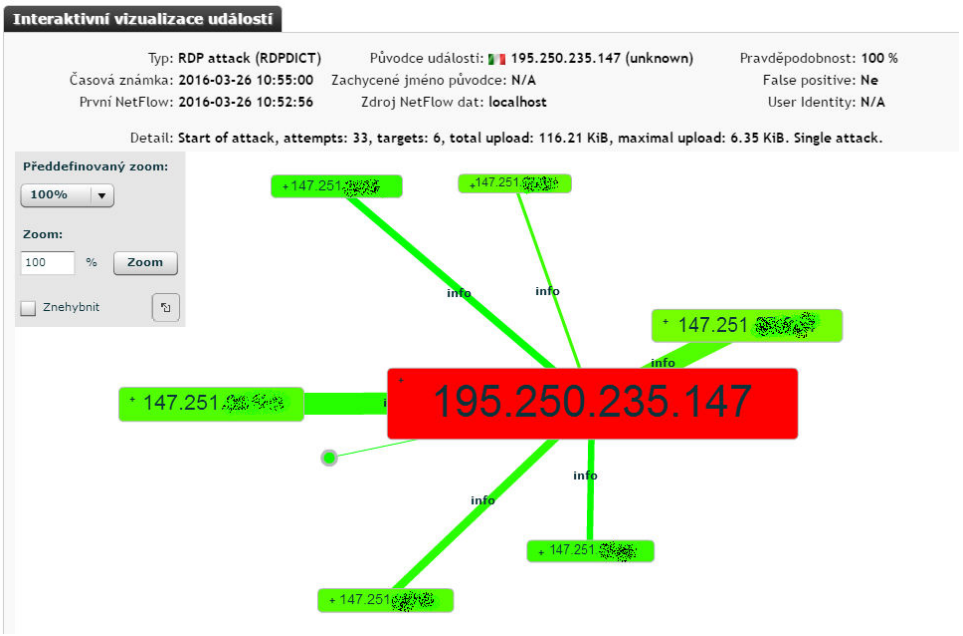
Každou z detekčních metod lze konfigurovat jednotlivě podle parametrů konkrétní detekce, přesto spolu všechny sdílí jednotné rozhraní pro ovládání. V terminologii Flowmon ADS je detekční metoda způsob odhalení útoku (příp. anomálie v provozu) a její konkrétní nastavení se nazývá instance metody. Díky tomu je možné spustit několik instancí stejné metody s různými parametry a chránit tak různé části sítě na různých úrovních.

Do společných parametrů všech instancí patří možnost aktivace/deaktivace, vytvoření/smazání instance, a nastavení doby uložení výstupů. Každá instance také potřebuje mít přiřazený zdroj flow dat, přičemž jsou

tyto zdroje zpracovávány samostatně, aby nedocházelo k ovlivnění metody případnou duplikací dat.

První skupinou detekčních metod jsou *běžné vzory chování sítě*, které generují události vždy na základě zpracování aktuální dávky flow dat (typicky 5 minutový interval dávek). Do této skupiny patří metody, které ke svému fungování nepotřebují znát znát širší kontext než je jeden datový interval. Patří sem například metody *BLACKLIST* a *HONEYPOT* pro detekci komunikace se „závadnými" adresami, detekce skenování portů (*SCANS*) nebo *REFLECTDOS* pro odhalení zneužití zranitelných NTP a DNS strojů k útokům. Seznam všech detekčních metod a jejich konfigurace je dostupná v uživatelské příručce [15].

Rozšíření některých metod detekce běžných vzorů pro SIP (Session Initiation Protocol) komunikaci tvoří druhou skupinu a obsahuje například detekce skenů nebo zahlcení SIP stanic (SIP flood).



Obrázek 2.2: Vizualizace zachyceného útoku ve Flowmon ADS 7.03.04

Ve skupině *pokročilých vzorů chování sítě* se detekují dlouhodobé trendy chování sítě na základe průběžného zpracování flow dat a kromě aktuální dávky flow dat má na výstup těchto metod vliv i předchozí chování jednotlivých účastníků komunikace.

Představitelem pokročilých metod detekce je *RDPDICT* k odhalování

útoků na autentizaci protokolu RPD. V základní konfiguraci metoda filtruje pouze komunikaci na portu TCP 3389, ale pomocí parametru *ObscurePorts* je možné tento filtr rozšířit na další, méně využívané porty. Po základním přefiltrování dat si metoda začne budovat strom útočníků a obětí a v případě pokusu o přihlášení se mezi nimi vytvoří hrana, která je přítomná po dobu definovanou parametrem *TimeWindow*. Grafické znázornění takto budovaného stromu je zobrazeno va obrázku 2.2.

Pokud se v tomto stromu vyskytne více pokusů o přihlášení od útočníka na oběť (parametr *AttackAttempts*), je vygenerována událost a je možné útočníka zablokovat dříve než uhodne správné heslo. Díky stromové struktuře je možné detekovat i distribuované útoky, kdy se útočník snaží schovat za velké množství IP adres, aby nepřekročil limity běžných detekcí. V tomto případě je událost vygenerována při překročení limitu přihlášení od jednoho útočníka definovaného násobkem parametrů *PartOfAttack* (procentuální vyjádření počtu pokusů z jedné adresy proti všech pokusům) a *AttackAttempts*. Metodu je možné upravit ještě nastavením minimálního počtu cílů útoku *MinTargets* nebo nastavit detekci pouze neúspěšných připojení ukončených TCP příznakem RST.

Zbylé dvě skupiny detekčních metod zahrnují *odvozené vzory chování*, které využívají výstupy předchozích detekčních metod a jsou spouštěny periodicky každou hodinu. Metody *obecné detekce anomálií* představují poslední skupinu a jsou založené na predikci chování síťového provozu. Na základě krátké historie je budována předpověď chování a pokud se predikovaná a reálná hodnota výrazně liší, je vytvořena událost obsahující pravděpodobného původce této anomálie.

Kapitola 3

Proudové zpracování dat

Klasický pohled na zpracování dat je jednoduchý. Data vzniknou, uloží se a v případě potřeby jsou dále zpracována. Takový přístup je vhodný, pokud dat není příliš mnoho, protože zpracovávající proces má k dispozici všechna data a může nad nimi dělat prakticky libovolné operace.

V současné době však tento přístup přestává vyhovovat jak z hlediska výpočetního výkonu tak doby, po kterou je nutno čekat na výsledky zpracování dat. V této kapitole se zaměřím na popis nového paradigmatu proudového zpracování dat a jeho rozdíly oproti klasickému dávkovému. Poté si představíme nejpoužívanější metody a systémy pro tvorbu datového proudu.

3.1 Odlišnosti proudového paradigmatu

Model datového proudu [6] je založen na principu, kdy data nejsou uložena na disku nebo v paměti, ale přicházejí (typicky po síti) v reálném čase tak, jak vznikají na zdroji dat. To s sebou přináší nové problémy při zpracování, které je nutné vyřešit:

- Řazení dat – zpracovávající systém nemá obvykle žádnou kontrolu nad tím, v jakém pořadí data přicházejí v rámci jednoho proudu nebo při použití více zdrojových proudů.
- Velikost dat – během proudového zpracování není dopředu známo, jaký objem dat bude potřeba zpracovat, přičemž teoreticky může být datový proud dokonce velikostně neomezen.
- Zpracování historických dat – jakmile jsou data jednou zpracována, jsou obvykle archivována nebo zahozena. V obou případech je opětovné zpracování části dat velmi komplikované a systém musí počítat s možností, že takové zpracování nebude možné vůbec.

Popsané odlišnosti proudového paradigmatu implikují také posun v přístupu k analýze dat. Klasický pohled využívá jednorázové dotazy,

které jsou spuštěny v určitý časový okamžik a zpracují všechna dostupná data. Tento přístup se dá zreplikovat i při proudovém zpracování, ovšem je k tomu potřeba vytvořit snapshot dat a zaměřit se pouze na něj. Opakem jsou průběžné dotazy. Výsledek průběžného dotazu je totiž upravován podle aktuálních dat přicházejících z proudu a je proměnný v čase.

Druhé možné dělení dotazů je na předdefinované a ad hoc dotazy. Pro proudové zpracování je typicky potřeba všechny dotazy nadefinovat před spuštěním systému kvůli optimalizaci a nemožnosti přepočítat stará data.

3.1.1 Posuvná okna

Problém s nemožností znovu zpracovat starší data řeší zavedení tzv. posuvných oken (sliding windows). Pro správnost výsledku obvykle není třeba přepočítávat všechna data od počátku měření, ale stačí pouze určitý omezený interval do minulosti. Tímto principem se inspirovala technika vytváření oken, kdy si systém proudového zpracování drží v paměti předem definované množství dat, nad kterými provádí výpočty.

Časové okno

Prvním typem posuvných oken jsou okna časová. Kromě toho, jak dlouho se data uchovávají (šířka okna), časová okna určují interval, jak často se má aktualizovat výsledek. Pomocí těchto dvou parametrů můžeme nastavit například, aby se braly v úvahu data za posledních 10 minut, ale výsledek se přepočítával každou minutu.

Pokud se obnovovací interval nastaví na velmi krátkou dobu, je možné touto technikou dosáhnout zpracování dat v téměř reálném čas, kdy zpoždění analýzy bude minimální. Nastavením obou parametrů časového okna na shodnou hodnotu lze nasimulovat i dávkové chování detekcí v systémech NfSen a Flowmon ADS (každých 5 minut se přepočítá okno o šířce 5 minut).

Count okno

Speciálním případem posuvného okna je nehledět na čas vzniku dat, ale pouze na počet příchozích zpráv z datového proudu, čímž se dosáhne konstantní velikosti dat pro zpracování

3.1.2 Vzorkování a dávkování

Pro urychlení výpočtů při proudovém zpracování některé systémy pracují s technikou shlukování zpráv do malých dávek (micro batch). Díky tomu dosahují vyšší efektivity během přijímání zpráv, jejichž doručení se obvykle potvrzuje vysílači (ack zprávy) a v případě velkého množství malých zpráv

dochází k zahlcení stroje pouze vytvářením a odesíláním ack paketů. Tyto malé dávky dat obvykle obsahují naakumulované zprávy za 1 vteřinu a nedochází tak k výrazným prodlevám mezi přijetím zprávy a výstupem analýzy.

Druhou technikou pro zvýšení výkonu je vzorkování zpráv, kdy dochází k zahazování každé n-té zprávy a tím pádem ke snížení výpočetních nároků. Výsledky analýzy založené na vzorkovaných datech jsou ovšem pouze aproximativní a pro účely bezpečnostních analýz tedy nevhodné.

3.2 Zdroje dat

V kontextu proudového zpracování dat není důležité, kde data vznikají, ani jaké informace obsahují. Pro jejich zpracování je podstatná forma přenosu od zdroje do systému pro zpracování.

Typické příklady zdrojů dat [18] reprezentují především senzorové sítě, monitorování síťového provozu nebo události z provozu webových stránek. Společnou charakteristikou všech zdrojů je nepřetržitá produkce velkého množství dat z distribuovaného prostředí. Součástí systému pro zpracování tedy musí být komponenta, která sjednotí příjem dat ze všech částí sítě.

Nejjednodušší možností doručení dat je vytvoření síťového spojení pomocí TCP soketu přímo do systému pro zpracování. I když je tato metoda přímočará, její praktické použití je limitováno, protože vyžaduje konfiguraci producentů dat pro každý zpracovávající systém zvlášť a v případě výpadku jsou data nenávratně ztracena.

Řešením těchto problémů je použití distribuovaného souborového systému HDFS (Hadoop Distributed File System) [5]. HDSF zajišťuje perzistenci dat v případě výpadku jednoho z disků, vyrovnávání zátěže mezi disky, integritu dat a možnost vytvořit zálohu celého systému. Takto uložená data jsou pak přístupná pro další zpracování ať už dávkově nebo pomocí proudového klienta.

Nejpokročilejší variantou odesílání dat ke zpracování jsou distribuované systémy pro zasílání zpráv. Podrobně si představíme systém Apache Kafka, který je využíván dále v práci.

3.2.1 Apache Kafka

Apache Kafka [21] je distribuovaný publish-subcribe systém pro zasílání zpráv. Zdroj dat se nazývá producent a pro jeho zapojení do systému je

potřeba použít speciálního klienta. V současné verzi 0.8.x nabízí projekt Kafka dohromady 18 druhů klientů pro různé systémy a programovací jazyky.

Základní datovou jednotkou v Kafce je zpráva, která je součástí tématu (topic). Témata je dále možno rozdělit na menší jednotky zvané oddíly (partitions), které představují seřazenou neměnnou posloupnost příchozích zpráv. O rozdělení dat do jednotlivých témat a oddílů se stará vždy producent, Kafka zprávě pouze přidělí jednoznačný identifikátor určující pořadí, v jakém zprávy do oddílu přišly.

Na druhé straně Kafky stojí příjemce zpráv, tzv. konzument. Výhodou členění dat do oddílů je, že konzument si může vybrat pro zpracování pouze část dat aniž by byl zaplaven celým objemem. Čtení probíhá na základě offsetu, což je identifikátor poslední přečtené zprávy z oddílu a na jeho základě konzument získává dosud nepřečtené zprávy. Tento systém umožňuje připojit více konzumentů ke stejnému oddílu a v případě výpadku jednoho z nich mu zaslat jím nezpracovaná data.

Architektura Kafky je založena na clusteru serverů (případně pouze jednoho) nazývaných brokery, kdy každý broker ukládá data z celého oddílu. Pro zvýšení výkonu a zajištění vyšší dostupnosti je možné nastavit replikační faktor, který způsobí, že oddíl bude nakopírovaný na více serverů

Kapitola 4

Systémy pro zpracování dat v reálném čase

V této kapitole se zaměřím na tři v současnosti nejpoužívanější frameworky pro proudové zpracování dat – jsou jimi Apache Samza, Apache Spark a Apache Storm. Tyto systémy budou v dalších částech práce testovány z hlediska výkonu při zpracování NetFlow dat a Apache Samza bude použita pro implementaci detekčního mechanismu útoku proti autentizaci RDP.

4.1 Apache Samza

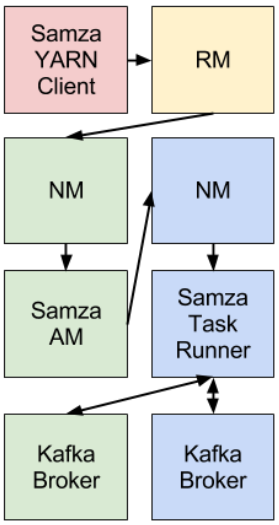
Společnost LinkedIn pro potřeby zpracování velkých dat (big data) v reálném čase bez prodlev způsobených dávkovým zpracováním vytvořila dva projekty. První, Apache Kafka, jako systém pro správu proudů zpráv je popsán výše v práci. Druhý systém od LinkedIn se nyní jmenuje Apache Samza [3] a byl navržen pro zpracování dat z Kafky s cílem proudového zpracování velkých objemů dat při zachování latence v řádu zlomků sekund [25]. Oba systémy byly vyvíjeny jako open source a po přechodu pod Apache Software Foundation jsou dále rozvíjeny touto komunitou.

Protože byl Apache Samza původně navrhnut pro zpracování dat z Kafky, je pro spolupráci s ní silně optimalizován a základní architektonické prvky vychází ze způsobu rozdělení dat do jednotlivých proudů a oddílů. Od tohoto modelu se odvíjí i rozložení práce na jednotlivé výpočetní stroje a charakter zpracování příchozích zpráv.

4.1.1 Správa clusteru

I když může Samza běžet pouze na jednom stroji, pro zpracování velkých dat je typické nasazení systému do výpočetního clusteru. Správu všech strojů v distribuovaném prostředí přenechává Samza systému Hadoop YARN (Yet Another Resource Negotiator) [2], který se stará o monitorování fyzických strojů a poskytuje nad nimi abstrakci pro spouštění libovolného kódu bez nutnosti znalosti konkrétního clusteru.

Hadoop YARN je systém vyvýjený rovněž společností Apache a jeho architektura je založená na principu master-slave, kdy je v clusteru vybraný jeden hlavní stroj, který dále rozděluje práci mezi podřízené stroje. Tento hlavní stroj, *Resource manager* (RM), si udržuje přehled o všech strojích v clusteru, zejména o počtu procesorů a celkové operační paměti, a jak velké množství těchto zdrojů je momentálně využito. Druhým cílem resource managera je plánování výpočetních úloh na podřízené stroje, přičemž zajišťuje rovnoměrné rozložení práce, správu výpadků (fault tolerance), logování, bezpečnost a oddělení zdrojů (resource isolation).



Obrázek 4.1: Komponenty clusteru [3]

Na každém stroji výpočetního clusteru je poté spuštěn Node Manager (NM), který se stará především o správu logických jednotek paralelizace tzv. kontejnerů. Kontejner je prostředí pro spuštění samotné aplikace a pro tyto účely si na stroji fyzicky rezervuje procesor (případně více jader) a operační paměť. Na jednom stroji tedy může běžet paralelně více kontejnerů a node manager řídí jejich tvorbu a zprostředkovává jejich přiřazení další komponentě zvané Application master (AM).

Správu aplikací v clusteru má na starosti Application Master. Ten registruje požadavky na spuštění aplikačního kódu a zajišťuje její spuštění. Pro tyto účely komunikuje s ostatními komponentami, získává kontejnery pro její běh a obsluhuje řešení chyb při pádu kontejneru.

Tuto architekturu systému přebírá i Samza při pohledu na dělení práce. Celý program nasazený do Samzy se nazývá práce (job) a specifikuje, jaké operace mají nad daty probíhat. Terminologií proudového zpracování dat tedy definuje transformaci vstupního proudu na proud výstupní. Práce je dále rozdělena na jednotlivé úlohy (tasks), které jsou spouštěny v YARN kontejnerech, přičemž jeden kontejner může vykonávat více úloh, ale jedna úloha nemůže být rozdělena na více kontejnerů (tedy ani více strojů).

Integraci Samzy s prostředím Hadoop YARN zajišťuje Samza YARN Client a Samza Task Runner. Tyto komponenty slouží pro komunikaci s managery clusteru a kontrolují celý běh aplikace od jejího nahrání na cluster, alokování potřebných zdrojů až po samotné distribuované spuštění na výpočetních strojích. Na obrázku 4.1 je znázorněno propojení všech zmíněných komponent a jejich rozmístění na strojích, kde každý stroj je reprezentován jinou barvou.

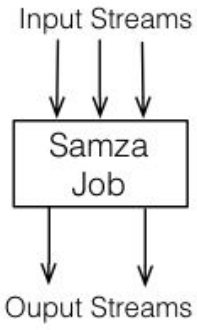
4.1.2 Datový a výpočetní model

Samza přebírá model datového proudu rozčleněného do témat a každá aplikace (job) tak může definovat, jaká témata bude zpracovávat. Pro každý oddíl vstupního proudu je při spuštění vytvořena vlastní úloha, čímž je zajištěno, že data z oddílu budou zpracována na stejném stroji a odpadá tak potřeba duplikace dat. Toto rozdělení znázorňují obrázky 4.2 a 4.3.

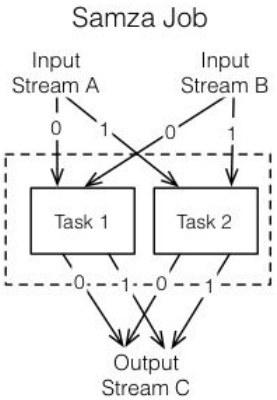
Přiřazení úlohy k oddílu je neměnné a v případě výpadku stroje je daná úloha spuštěna jinde se stejným přiřazeným oddílem. Jelikož úloha nemůže zpracovávat více oddílů z jednoho tématu, je nutné určit stupeň paralelizace systému už během vytváření datového proudu v Apache Kafka. Samza toto rozdělení přebírá a nemůže ke zpracování využít více strojů (resp. kontejnerů) než jaký je celkový počet oddílů ve vstupních tématech.

Základní datovou jednotkou je zpráva (message), která je přiřazena do konkrétního oddílu. Na rozdělení zpráv do oddílů opět Samza nemá žádný vliv a zodpovědnost za korektní rozdělení má producent dat. Po přijetí zprávy úlohou dojde k jejímu zpracování. Výsledkem může být cokoliv od aktualizace databáze, odeslání mailu až po vytvoření datového proudu. Právě tvorba nového proudu obsahující zprávy s výsledky umožňuje propojit jednotlivé stroje clusteru tak, aby se náročné operace počítaly postupně a zátěž se rozložila rovnoměrně. Tím je možno vytvořit libovolný graf výpočtu, v praxi obvykle acyklický, který určuje jaké mezivýsledky budou počítány a jaké úlohy je budou zpracovávat.

Všechny úlohy jsou navzájem izolované a nemohou data přímo sdílet.



Obrázek 4.2: Koncept zpracování proudů [3]



Obrázek 4.3: Rozdělení práce na úlohy [3]

Pokud je cílem výpočtu získat výsledek závisející na datech z celého tématu (např. celkový počet zpráv za minutu), je nutné pro takový výpočet vytvořit graf výpočtu, ve kterém se mezivýsledky z jednotlivých oddílů slévají do jednoho proudu zpracovávaného samostatným strojem. Apache Samza nenabízí žádné sdílené proměnné a vynucuje tak důsledné používání proudového paradigmatu.

4.2 Apache Spark

Druhým zástupcem systémů pro proudové zpracování dat je Apache Spark [28]. Tento systém byl původně vytvořen pro distribuované zpracování velkých dat na základě Map reduce principu. Adaptaci na zpracování datových proudů přinesla až komponenta Spark Streaming, která poskytuje rozhraní mezi příjmem proudu a samotným zpracováním ve Spark Engine.

Kvůli tomuto rozdělení je zpracování dat ve Sparku na pomezí mezi proudovým a dávkovým modelem. Příchozí zprávy totiž nejsou zpracovávány tak jak přicházejí, ale Spark Streaming z nich nejprve vytváří malé dávky zvané RDD (Resilient Distributed Dataset), jak je znázorněno na obrázku 4.4. Dávkování vstupního proudu probíhá obvykle ve velmi krátkých intervalech (1 sekunda) a je tak někdy nazýván jako diskretizovaný proud, zpracování je poté nazýváno jako téměř v reálném čase (near real-time).

Obrázek 4.4: Tvorba dávek z vstupního proudu [4]

Pro správu výpočetního clusteru nabízí Spark hned několik možností zahrnující vlastní správu (standalone mód) a dedikované systémy Hadoop YARN, Apache Mesos a Amazon EC2. Výsledná architektura je poté velmi podobná Samze – o správu všech strojů se stará Cluster Manager (ve standalone módu zvaný master node) a jednotlivé stroje spravuje Worker. Spouštění aplikací a alokaci fyzických zdrojů má na starosti komponenta Executor, ve kterém běží jednotlivé úlohy výpočtů.

Spark má ovšem navíc jednu komponentu zvanou Spark Context, která umožňuje přímou komunikaci výpočetních uzlů pomocí dvou typů sdílených proměnných. Prvním z nich je *broadcast* proměnná určená pro sdílení velkých objektů pouze pro čtení. S její pomocí Spark Context zajistí, že každý worker dostane stejnou kopii dat a že ji obdrží pouze jednou. Druhým typem je tzv. akumulátor, který naopak slouží pouze pro zápis a je typicky využíván pro implementaci čítačů a paralelních součtů. SparkContext navíc určuje rozdělení práce mezi workery a zajišťuje doručení vstupních dat ke zpracování.

Výpočetní model je vychází z původního návrhu MapReduce, ale obsahuje vlastní rozšíření pro urychlení zpracování. Vstupní data nejprve prochází transformací, kdy se na každý prvek RDD aplikuje daná funkce, je provedeno filtrování nebo sdružení dat podle určitého klíče. Výstupem transformace je modifikované RDD vstupující do další fáze zpracování zvané akce. Akcí jsou především vlastní agregační funkce a součty přes definovaný prvek, speciálním případem je potom funkce *foreach*, která umožní vykonat libovolnou akci pro každý příchozí balík dat zvlášť.

4.3 Apache Storm

Poslední z vybraných frameworků, Apache Storm [1], je zástupcem čistě proudového zpracování dat. Oproti předchozím systémům dává Storm uživateli naprostou svobodu v použitých funkcích i rozložení práce mezi stroje.

O správu clusteru se obvykle stará opět Hadoop YARN, ovšem o přiřazení zdrojů jednotlivým úlohám rozhoduje samotná aplikace tvorbou *topologie*. Ta rozděluje instance úlohy podle funkcionality na *Spout* a *Bolt*. Úlohou spoutu je napojit se na zdrojový proud dat a každou příchozí zprávu opatřit unikátním identifikátorem a označit, ze kterého proudu přichází. Tím vznikne základní datová jednotka zvaná *tuple* dále zpracovávaná v boltech.

Během tvorby topologie je specifikován přesný počet spoutů a boltů a pro každý je definováno, kolik výpočetních zdrojů mu bude přiděleno (především počet jader procesoru a paměť). Zároveň s tím je definováno i propojení strojů datovými proudy a vzniká tak libovolný graf výpočtu.

Komunikace jednotlivých úloh aplikace je tedy plně závislá na topologii a sdílení dat je možné pouze v rámci jednoho workeru (resp. stroje). Nevýhodou Stormu v tomto směru je povinné potvrzování doručení všech zpráv (tuple) odeslaných mezi workery, což výrazně snižuje výkon při zpracování velkého množství malých zpráv.

Srovnání popsaných systémů

Celkové srovnání názvosloví a použitých konceptů v popisovaných nástrojích je shrnuto v tabulce 4.1. Kromě vlastností popsaných výše tabulka zahrnuje i přehled programovacích jazyků, které je možné použít pro vývoj, a podpora systému pro tvorbu oken nad proudem dat, která bude zásadní v praktické části práce a bude detailně popsána v rámci implementace testovací metod.

	Samza	**Storm**	**Spark**
Příjem dat z proudu	Consumer	Spout	Receiver
Datová jednotka	Zpráva	Tuple	RDD
Správa clusteru	YARN	YARN	Standalone, YARN, Mesos
Paralelizace	Dle oddílů proudu	Určuje topologie	Určuje SparkContext
Zpracování zpráv	Sekvenční po jednom	Sekvenční po jednom	Tvorba dávek
Sdílení dat mezi stroji	Databáze, vlastní implementace komunikace	Databáze, vlastní implementace komunikace	SparkContext akumulátory broadcast proměnné
Programovací jazyk	Java, Scala	Java, Clojure, Scala, libovolný přes JSON API	Java, Scala, Python
Časová okna	Proprietární	Vlastní implementace Spoutu	Proprietární
Count okna	Vlastní implementace	Vlastní implementace Spoutu	Pomocí akumulátoru

Tabulka 4.1: Srovnání jednotlivých systémů

Kapitola 5

Výkonnostní testování vybraných frameworků

Cílem praktické části mé práce bylo vytvoření sady testů pro systém Apache Samza a následné porovnání jeho výkonu na různých konfiguracích výpočetního clusteru. Tato testovací sada byla vytvořena v rámci výzkumného projektu *TA04010062/2014 Technologie pro zpracování a analýzu síťových dat velkého rozsahu* a stejné metody byly implementovány v systémech Apache Spark a Storm, což umožnilo následné srovnání výkonu i napříč těmito frameworky. Výsledky tohoto srovnání byly publikovány v článku *A Performance Benchmark for NetFlow Data Analysis on Distributed Stream Processing Systems* [13].

Protože zmíněné systémy používají odlišné principy v konkrétních detailech implementace, byly testy uzpůsobeny tak, aby byly shodné pro všechny systémy. Stejně tak byla zvolena celá architektura výkonnostních testů, aby byly podmínky pro všechny frameworky stejné.

Výkonnostní test zaměřený přímo na srovnání systémů proudového zpracování dat byl představen v článku StreamBench [23], ovšem jeho výsledky ukazují na velké výkyvy výkonnosti jednotlivých systémů při změnách velikosti a obsahu příchozích zpráv v proudu. Pro bezpečnostní monitorování sítě jsou však zpracovávána pouze Netflow/IPFIX data, která mají pevně danou strukturu a velikost zpráv je přibližně konstantní.

5.1 Testovací scénáře

Pro výkonnostní testování bylo navrženo celkem šest scénářů. První z nich slouží pouze pro otestování základního výkonu systému, ostatní poté reprezentují různě složité operace běžně používané v detekcích síťových útoků.

- **Žádná operace** – základní scénář, ve kterém framework pouze přijímá příchozí data a nijak je nezpracovává. Pro zapojení do testů je potřeba počítat pouze celkový počet zpráv a po průchodu definovaného počtu odeslat jednu zprávu uzavírající test.

- **Filtrování** – každá detekční metoda pro svoji funkci potřebuje vybrat pouze toky, které jsou pro ni významné. Tuto funkcionalitu simuluje filtrační test, kdy je příchozí zpráva zpracována, je z ní vyextrahována zdrojová IP adresa toku a je inkrementován čítač detekovaných toků.
- **Suma** – rozšířením filtrace je suma definované položky předfiltovaných toků. V našem testování počítáme celkový počet paketů pro danou zdrojovou adresu.
- **Agregace** – další stupeň výpočetní a paměťové náročnosti je počítání paketů pro každou zdrojovou adresu. Během zpracování tak nestačí pouze jeden sdílený čítač, ale je nutno synchronizovat mnohem větší datovou strukturu mezi všemi stroji clusteru.
- **Top N** – v tomto scénáři musí framework nejprve provést agregaci nad celou datovou sadou a výsledek seřadit sestupně podle počtu paketů. Výstupem je zpráva obsahující N adres s nejvíce pakety.
- **Sken** – simulace detekce skenování sítě pomocí SYN paketů, která kombinuje všechny výše popsané scénáře filtrováním paketů podle TCP příznaků, agregací počtu toků přes všechny IP adresy a následného nahlášení 100 nejvíce skenujících adres.

Sledovaným parametrem během výkonnostních testů je propustnost frameworku měřená v počtu zpracovaných zpráv za vteřinu. Další neméně významnou vlastnost představuje korektnost výsledku, kdy jsou správné výsledky na dané datové sadě vypočítány dopředu a výstup každého testu je vůči nim porovnán. Pro bezpečnostní analýzu je totiž důležité zpracovat veškerá data a zahazování zpráv nebo samplování z důvodu přetížení je proto nepřijatelné.

5.2 Architektura výkonnostního testování

5.2.1 Testovací data

Data použitá pro výkonnostní testy pochází z páteřní linky v Chicagu, která pravidelně zveřejňuje sdružení CAIDA pro výzkumné účely [7]. Použitý vzorek pochází z roku 2015, a protože je k dispozici ve formě kompletního PCAPu, byl nejprve transformován do podoby Netflow, kde je každý tok reprezentován jako JSON záznam.

Z takto upraveného balíku dat byl následně extrahován první milion toků s cílovou IP adresou 50.224.90.224, který posloužil jako základní dataset. Tento milion toků byl poté opakované vkládán do výsledného datasetu

tak, že při každé iteraci došlo ke změně cílové adresy, díky čemuž bylo snadné rozdělit toky do oddílů proudu. Vzorový tok vypadá následovně:

```
{"date_first_seen":"2015-07-18T18:07:33.475+01:00",
 "date_last_seen":"2015-07-18T18:07:33.475+01:00",
 "duration":0.000,"src_ip_addr":"86.135.210.175",
 "dst_ip_addr":"31.157.1.1","src_port":54700,
 "dst_port":80,"protocol":6,"flags":".A....",
 "tos":0,"packets":1,"bytes":56}
```

Pro měření propustnosti frameworků je důležitá velikost jednotlivých zpráv. V našem případě je velikost téměř konstantní (liší se pouze délka IP adresy, která je v JSONu přítomná jako textový řetězec), v průměru 270 bajtů na zprávu.

5.2.2 Hardware a propojení strojů

Testování frameworků probíhalo na clusteru složeného ze sedmi VMware vSphere 6.0 strojů s následujícími parametry každý z nich:

- 2 x Intel® Xeon® E5-2670 (16/32 HT cores in total),
- 192 GB 1600M MHz RDIMM ECC RAM,
- 2 x HDD 600 GB SAS 10k RPM, 2,5”(RAID1),
- 10 Gbit/s síťové propojení, 1 Gbit/s virtual NICs.

Z těchto strojů byly dva vyhrazeny pro běh Apache Kafka, který poskytoval data testovaným systémům, zbylé stroje sloužily pro spuštění testovaných systémů. Přehled použitých verzí software je uveden níže.

- Debian Linux 8.1.0 x64
- Oracle Java 1.8.0
- Scala 2.9.2
- Apache Hadoop 2.7.1
- Apache Zookeeper 3.4.5
- Apache Kafka 0.8.2.1
- Apache Spark 1.4.1
- Apache Storm 0.9.4
- Apache Samza 0.8.0

Použitá virtualizace umožnila vytvořit různé konfigurace strojů a změřit tak rozdíly výkonu v prostředí s výrazně omezenou výpočetní kapacitou až po cluster složený z velmi výkonných strojů. Pro výkonnostní testování tedy byly připraveny čtyři varianty strojů popsané v tabulce 5.1.

Konfigurace	Počet jader procesoru	Operační paměť	Velikost disku
vm_large	32	128 GB	300 GB
vm_normal	16	64 GB	300 GB
vm_medium	8	32 GB	300 GB
vm_small	4	16 GB	300 GB

Tabulka 5.1: Konfigurace virtuálních strojů pro testování.

Apache Kafka běžel vždy na nejvýkonnější konfiguraci, aby nedocházelo k omezení výkonu právě tímto systémem.

Z hlediska výpočetního clusteru už dále nedocházelo k žádným zásahům do probíhajícího testování. Všechny frameworky dostaly k dispozici shodné výchozí podmínky a efektivita správy strojů jednotlivými systémy se tak promítla do výsledků testu.

5.2.3 Architektura testů

Kromě shodného hardware a software pro testy bylo nutné zajistit i stejné prostředí pro zasílání zpráv a měření výkonu.

Všechny testované frameworky přijímaly data z tématu *tst* Apache Kafky, který byl rozdělen na 100 oddílů a každý oddíl obsahoval přesně 1 milion toků, kde všechny toky měly v rámci oddílu stejnou dst IP adresu. Vzhledem ke struktuře dat popsané výše bylo třeba při každém testu odeslat po síti a zpracovat přibližně 27 GB dat. Pro testy byl rovněž nastaven *partitioning* dat na 2, čímž bylo umožněno číst data z obou strojů Kafky zároveň.

Původní myšlenka využití Kafky jako zdroje dat byla, že se téma naplní pouze jednou a frameworky pro každý test vyresetují offset tématu na nulu a budou tak číst data opět od začátku. Ovšem později se ukázalo, že frameworky potřebují v celku dlouhý čas na inicializaci clusteru a zahájení výpočtů na všech strojích. Ve výše popsané konfiguraci pozorovaná prodleva dosahovala až jedné minuty a stroj inicializovaný jako první začal okamžitě zpracovávat data uložená v tématu, což výrazně ovlivnilo výsledky testu.

Systém proudového zpracování dat by však při reálném nasazení měl pracovat nepřetržitě v řádech týdnů (měsíců) a pro bezpečnostní monitorování je tak důležitý výkon měřený čistě nad živými daty přicházejícími z proudu a čas potřebný pro inicializaci nástroje tak nehraje roli.

Z výše uvedených důvodů byla proto architektura testu změněna z vyčítání starých dat z tématu na zpracování dat přicházejících do Ka-

fky až během testu. To ovšem položilo nové nároky na Kafku, která nyní musela zvládat přijímat i odesílat data rychleji než je frameworky zpracují, aby testy neovlivňoval výkon clusteru pro Apache Kafka. Při testech se také ukázalo, že skripty pro plnění témat ani producenti dat volně šířené s Kafkou zdaleka nedosahují potřebného výkonu, a bylo tak nutné vytvořit vlastní producent s dostatečnou rychlostí.

Ve spolupráci s Danielem Tovarňákem byl proto v jazyce Erlang vytvořen nástroj *eKafSender*. Tento skript plní v testech dvě funkce – odesílání dat a měření času výpočtu. Pro dosažení potřebného výkonu odesílání dat je potřeba využít co nejvíce paralelizace, proto *eKafSender* vytváří 100 vláken, kde každé plní jeden oddíl tématu. Díky tomu je rychlost zápisu do Kafky prakticky omezena pouze čtením/zápisem na disk a bylo ověřeno, že celkový výkon tohoto nástroje a Kafky při všech testech převyšoval výkon frameworků zpracovávajících data.

Poslední otázkou při návrhu testů bylo měření samotné propustnosti. Aby byly výsledky korektní vůči všem frameworkům, není možné použít žádnou metriku počítanou přímo daným systémem. Funkcionalita měření proto musí být součástí externího systému, které celé testovací prostředí obaluje. Takové požadavky jednoduše splňuje nástroj *eKafSender*, který zaznamená timestamp prvního odeslaného toku. Po dokončení odesílání dat poté čeká na příchod speciální zprávy označující konec výpočtu (kterou frameworky jednotně zasílaní do připraveného tématu v Kafce) a opět zaznamená čas příchodu. Následně z celkového počtu odeslaných zpráv a rozdílu obou časových značek vypočítá průměrnou rychlost zpracování dat v tocích za vteřinu. Tyto výsledky jsou při testování ukládány do souboru pro usnadnění automatizace testů a pozdějšího zpracování výsledků.

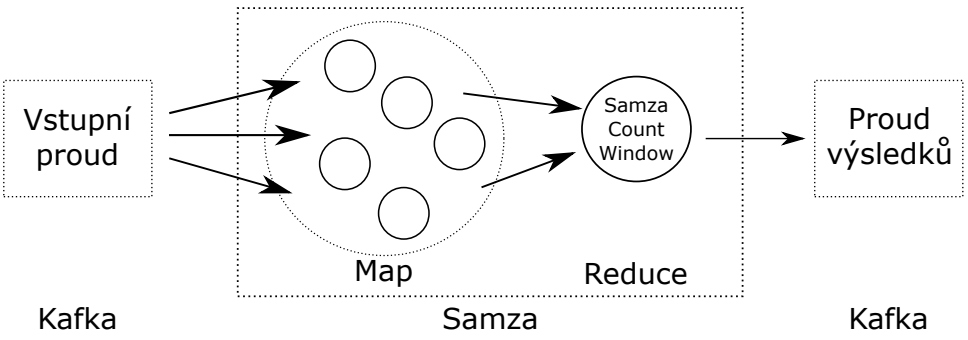
Stejně jako při exportování toků ze sondy na kolektor nedává smysl posílat každý tok v samostatném paketu, i v našem testovacím prostředí je vhodné toky zasílat z Kafky do frameworku ve větších balících a ušetřit tak šířku pásma pro skutečná data namísto hlaviček paketů. Protože všechny frameworky nativně podporují příjem dat z Kafky v dávkách, byl tento princip použit pro příjem dat. Po otestování různých velikostí dávek bylo rozhodnuto o použití velikosti jednoho tisíce toků v dávce, což pozitivně ovlivnilo výkon všech systémů při zachování běžné velikosti paketů a zpracování dat v reálném čase (při zpracování milionu toků za vteřinu činí prodleva dávky pouhou jednu milisekundu).

5.3 Implementace testovací metod v Apache Samza

Mým prvním úkolem v praktické části práce bylo zpracovat výše popsané výkonnostní testování a jeho metody v prostředí Apache Samza. Protože všechny testy jsou postavené okolo myšlenky count okna, které Samza nativně nepodporuje, bylo potřeba tento koncept naprogramovat manuálně.

Cílem bylo distribuovaně zpracovat 100 milionů příchozích zpráv a poté odeslat jednu zprávu do připraveného tématu. Vzhledem k velmi pevné vazbě Samzy na oddíly vstupního proudu, šlo systém nakonfigurovat tak, že každý *task* měl přesně určený počet toků ke zpracování. Ovšem stále zůstává problém, kdy každý task má svůj výsledek, který je potřeba dát dohromady s ostatními a vytvořit tak jeden společný.

Proto jsem navrhl architekturu systému podle myšlenky MapReduce, ve které první *job* obstarává zpracování příchozích zpráv a předpočítané výsledky odesílá do samostatného *jobu*, kde dochází k jejich agregaci (reduce) a také ke kontrole počtu zpráv v count okně. Toto schéma výpočetního modelu znázorňuje obrázek 5.1.



Obrázek 5.1: Architektura zpracování zpráv v testech

Ve všech testech je struktura prvního *jobu* velmi podobná, obsahující čítač zpráv a datovou strukturu pro uložení mezivýsledku. Nyní si testy detailně popíšeme.

Implementace testu **žádná operace** příchozí zprávu vůbec neprohlíží a jediná jeho operace je odeslání počtu přijatých zpráv v intervalech načtených z konfiguračního souboru.

Testovací scénáře **filtrování** a **suma** už příchozí zprávy zpracovávají. Nejprve je provedena deserializace JSON objektu pomocí knihovny Jackson (*com.fastermxl.jackson*) ve verzi 2.6.0-rc2, která je opět využívána ve všech testovaných systémech. Pro zpracování v Samze je nutno takovou zprávu namapovat na Java objekt. Proto jsem vytvořil POJO (Plain Old Java Ob-

jekt) třídu Flow, která všechny položky síťového toku reprezentuje pomocí základních datových typů a je tak možné s nimi dále pracovat.

V obou testech je následně z Flow objektu získána zdrojová IP adresa a porovnána s adresou 141.57.244.116. Při filtrování dojde pouze k inkrementaci čítače toků, v count je navíc z daného toku vyextrahován počet paketů, který se přičte do sumy.

Další dvojice podobných testů je **agregace** a **top n**, ve kterých jsou výsledky ukládány do struktury HashMap, kde jako klíč slouží řetězec obsahující zdrojovou IP adresu toku a hodnota je počet paketů pro tuto adresu. Protože první *job* počítá pouze distribuované mezivýsledky, musí top n zpracovávat a udržovat všechna data (řazení a zahození při odesílání mezivýsledku by teoreticky mohlo ovlivnit celkový výsledek). Kvůli tomu jsou pak implementace obou testů prakticky totožné. Do druhého *jobu* je mapa odeslána ve formě textového řetězce serializovaného na proud bajtů s využitím knihovny Jackson.

Poslední testovací metoda **sken** využívá stejné principy jako ostatní metody, pouze při zpracování POJO objektu porovnává více atributů a sledovaným parametrem je zde počet toků.

Druhý *job* nazvaný *SamzaCountWindow* má na starosti kompletování výsledků a implementaci count okna. Na jednom místě (resp. stroji/úloze) musí dokázat obsloužit všechny testovací metody. Proto zprávy s mezivýsledky obsahují identifikátor testu, počet toků použitých pro výpočet dané části výsledku a samotná data. Po rozparsování zprávy tedy *SamzaCountWindow* ví, kterému testu data patří a aktualizuje připravené datové struktury pro každý test. Spolu s tím je aktualizován i celkový počet zpracovaných toků.

V případě testů top n a sken *SamzaCountWindow* navíc provádí seřazení výsledků. Pro vyšší efektivitu tohoto finálního kroku zpracování byl vytvořen vlastní *komparátor* a datová struktura změněna na TreeMap, ve které jsou data řazena již při vkládání.

Pokud celkový počet zpracovaných toků dosáhne hodnoty definované konfigurací jako velikost count okna, je vygenerována zpráva do výstupního proudu obsahující kompletní výsledek daného testu, jsou vynulovány čítače a je uvolněna veškerá použitá paměť. Díky tomu má následující okno stejné výchozí podmínky a *job* tak může běžet nepřetržitě. Tato výsledná zpráva je poté zaznamenána nástrojem *eKafSender* a test se tím ukončí.

Pro správné fungování count okna je nutné v konfiguračních souborech nastavit počet zpráv použitých v jednom mezivýsledku tak, aby byla celková velikost okna tímto počtem dělitelná. V opačném případě by

docházelo k nepřesnostem při výpočtu propustnosti (který počítá s přesně definovaným počtem zpráv) a místo výsledku je proto odeslána chybová hláška signalizující špatné nastavení, případně fatální selhání Samzy. K takové události však během testování nedošlo a výsledky se vždy shodovaly s předpočítanými hodnotami.

5.4 Testování frameworku Apache Samza

Pro usnadnění testování a jeho automatizaci jsem připravil celkem tři skripty. Skript *install.sh* na všechny stroje clusteru stáhne a nainstaluje systém Hadoop 2.7.1 potřebný pro běh YARN cluster manažera. Poté provede konfiguraci YARNu podle předem nastaveného rozdělení na master a slave uzly. Aby mohla být Samza spuštěna na YARN clusteru, je potřeba mít v YARN přiložené knihovny samza-yarn_2.10-0.8.0 a samza-core_2.10-0.8.0, kromě nich musí být přítomny ještě doplňkové knihovny pro Scala a Slf4j. Všechny tyto knihovny jsou součástí instalačního balíčku a skript je rozdistribuuje na cluster.

Druhý skript, *prepareSamza.sh*, slouží pro kompilaci a konfiguraci testovacích metod. Protože Samza potřebuje mít uložené adresy YARN mastera a Kafka serverů v konfiguračích souborech, skript tyto adresy vloží do všech testů, čímž připraví celý systém ke spuštění.

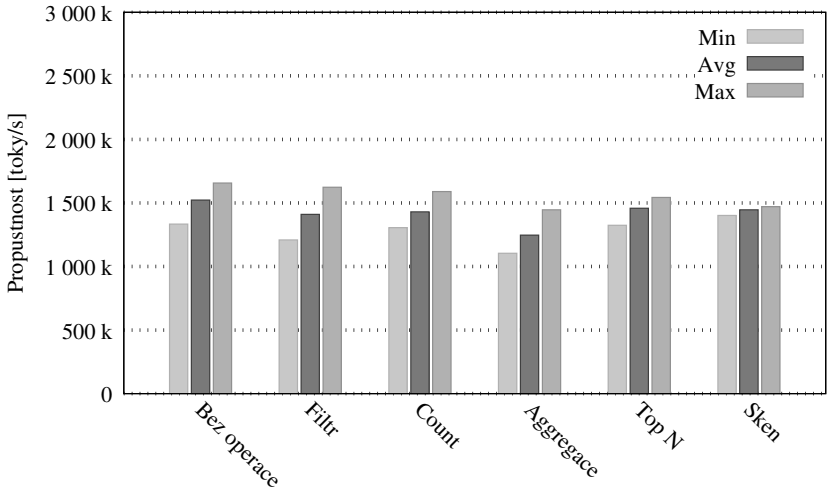
Skript *run_tests.sh* představuje automatizované spouštění testů. Procedura testu začíná znovuvytvořením všech používaných Kafka témat, aby každý běh testu měl stejné prostředí (kvůli správě témat Kafka vytváří velké množství dočasných souborů a objemy dat při testování můžou po čase způsobit zpomalení systému). Následuje spuštění *SamzaCountWindow* a dané testovací metody. Z důvodů inicializace úloh na jednotlivých strojích je nutné počkat, až se nastartuje všech 100 úloh. Dalším krokem je spuštění *eKafSenderu*, který naplní témata a připravená testovací metoda data zpracuje. Na závěr každého testu je restartován YARN, opět z důvodu shodných výchozích podmínek.

Každý test je spuštěn celkem 10x a záznam běhu (zpráva ukončující test a vypočítaná propustnost) je vždy uložen do dočasného souboru. Po dokončení všech testovacích metod skript stáhne tyto záznamy a uloží je do složky, odkud byl zavolán.

5.4.1 Výsledky testů

Celá testovací sada byla spuštěna na čtyrech různých konfiguracích strojů. V prvním testu byl pro běh framewroku vyhrazen pouze jeden *vm_large*

stroj s 16 procesorovými jádry. Výsledky jsou zobrazeny v grafu 5.2 a je z nich patrné, že Samza nemá problém fungovat na malém omezeném clusteru, kdy na jednom stroji běží YARN master i slave a je spuštěno 101 úloh najednou (100 pro každý oddíl testu a 1 pro count okno).

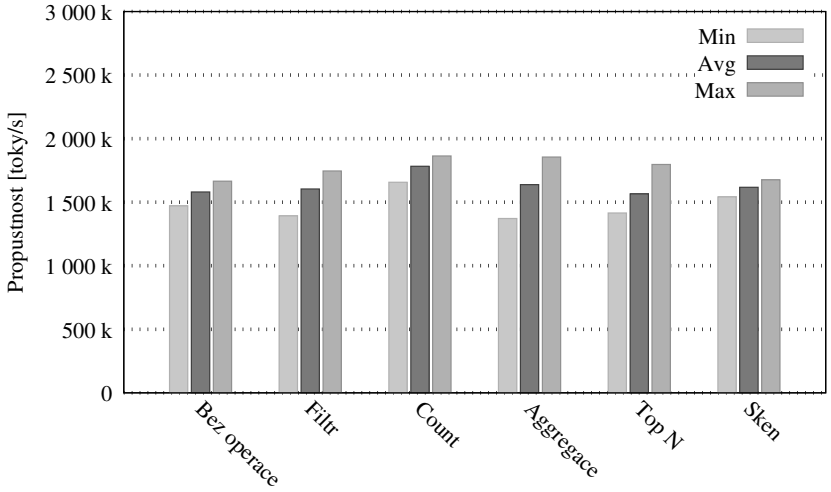


Obrázek 5.2: Výkon testovacích metod při použití 1x *vm_large* stroje.

Ve druhém scénáři byl omezen počet procesorových jader na polovinu (konfigurace *vm_normal*) pro otestování výkonu v ještě více omezených podmínkách. Naměřenou propustnost zobrazuje graf 5.3. Výkon oproti očekávání vzrostl a pohyboval se nad hranicí 1,5 milionu zpracovaných toků za vteřinu. Tento nárůst byl pravděpodobně způsoben rychlejším přepínáním úloh mezi menším počtem procesorů jednoho stroje způsobené menší režií přepínání v rámci jednoho fyzického procesoru oproti distribuci práce na více jednotek.

Optimalizaci Apache Samzy pro běh na na větším množství slabých strojů dokládají i zbylé provedené testy. Použití čtyř výpočetních uzlů typu *vm_medium* ukazuje graf 5.4. V této konfiguraci dokonce některé běhy testů překonaly rychlost 2 M toků za vteřinu a narazily na teoretický limit testů, jelikož stroje clusteru byly propojeny pouze 10 Gbs linkou, která nestíhala přenášet takový objem dat. Tuto domněnku potvrzuje i fakt, že jsem během těchto testů pozoroval vytížení procesorů pouze okolo 90 %, přičemž při ostatních testech všechny procesory běžely konstantně na plný výkon.

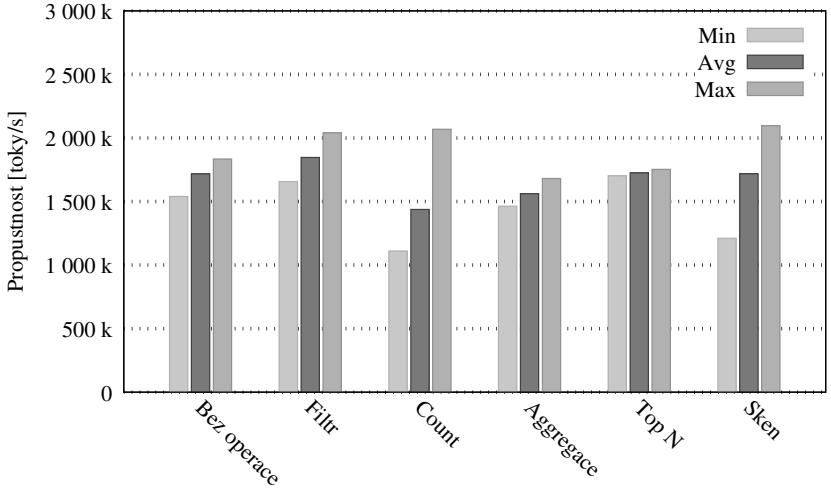
Jako poslední testovaný scénář byly použity 4 stroje typu *vm_small*, které výkonem odpovídají běžným domácím počítačům a lepším notebookům. I na takto výrazně oslabeném clusteru dokázala Samza udržet výkon přes



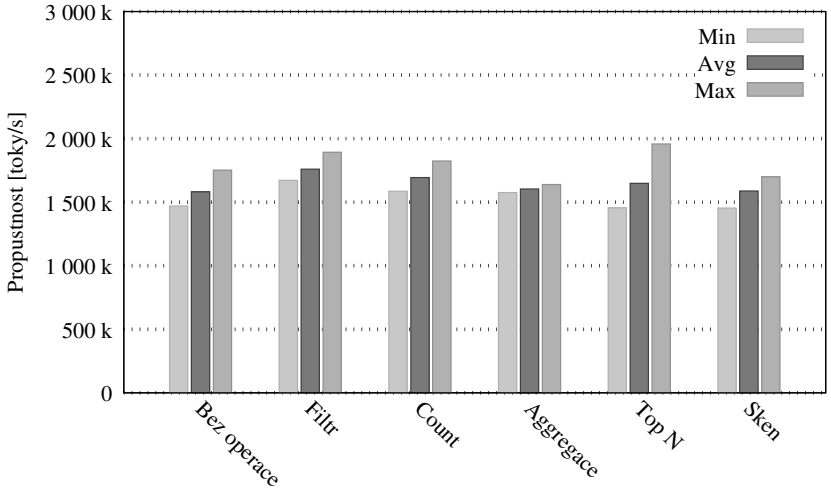
Obrázek 5.3: Výkon testovacích metod při použití 1x *vm_normal* stroje.

1,5 milionu toků za vteřinu a překonala výkon *vm_large* stroje.

Velmi zajímavým pozorovaným výsledkem je srovnání výkonu jednotlivých testovacích metod v rámci stejné konfigurace. Každá testovací metoda postupně přidává další funkcionalitu, kterou je nutné počítat pro každý příchozí tok a při daném datovém objemu by tento výpočet



Obrázek 5.4: Výkon testovacích metod při použití 4x *vm_medium* strojů.



Obrázek 5.5: Výkon testovacích metod při použití 4x *vm_small* strojů.

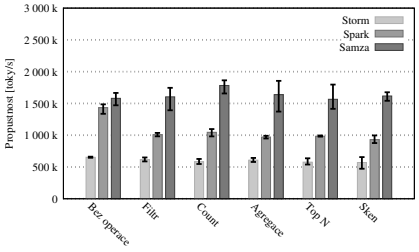
měl výrazně ovlivnit propustnost. Ovšem nasazení v reálném prostředí ukázalo, že v případě frameworku Apache Samza je výkon potřeba hlavně na správu clusteru a doručování/odesílání zpráv a jejich serializaci. Zvolené metody simulující jednotlivé kroky bezpečnostní analýzy provozu nebyly natolik výpočetně náročné, aby se jejich vliv výrazně promítl do naměřené propustnosti. Odchylky propustnosti testovaných metod jsou navíc ovlivněny náhodnými vlivy způsobenými plánováním procesů a pro jejich odstranění by bylo potřeba mít řádově vyšší počet opakování každého testu. Tyto odchylky jsou potom jasně patrné z velkých rozdílů mezi minimální a maximální hodnotou propustnosti a vyskytovaly se u všech testovaných frameworků. Bohužel jejich příčinu se nepodařilo spolehlivě objasnit, protože se objevovaly konzistentně ve všech konfiguracích clusteru a všech metodách, navíc byly pozorovány i na vývojovém clusteru. Při nasazení systémů proudového zpracování dat do praxe je tak nutné s tímto fenoménem počítat.

5.5 Porovnání s Apache Spark a Apache Storm

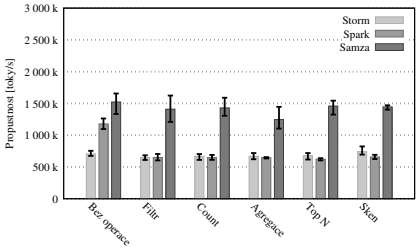
V rámci výzkumného projektu byly stejné testy implementovány ještě ve frameworcích Apache Spark a Apache Storm. V průběhu vývoje testovacích metod probíhala spolupráce všech týmů vyvíjejících testy pro jednotlivé systémy, aby byly implementační detaily co možná nejvíce podobné

a výsledky tak mohly být přímo srovnány.

Výsledky provedených testů zachycují grafy 5.6, 5.7, 5.8, 5.9. Pro přehlednost je zde zachycen pouze průměrný výkon v daném testu a minimální, maximální naměřené hodnoty jsou zobrazeny pomocí vymezení chyby (error bar).



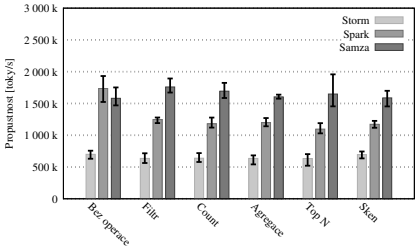
Obrázek 5.6: Srovnání výkonu na konfiguraci 1x *vm_normal*.



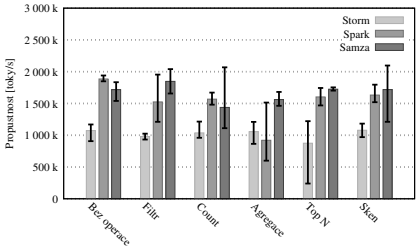
Obrázek 5.7: Srovnání výkonu na konfiguraci 1x *vm_large*.

Z výsledků je patrné, že všechny testované frameworky byly schopné zpracovat alespoň 500k toků za vteřinu i v pro ně nejméně vhodné konfiguraci výpočetního clusteru. Tím s přehledem splnily stanovené požadavky na bezpečnostní analýzu v síti národního rozsahu [13], ve které je potřeba kontinuálně zpracovávat každou vteřinu přibližně 100 tisíc toků s výkyvy ve špičce nebo během útoků až k hranici 300k toků za vteřinu.

Pozorovaná optimalizace Samzy pro běh na více slabších strojích je u ostatních frameworků mnohem výraznější a především Apache Spark vykazuje obrovské rozdíly výkonu v závislosti na použitých strojích. Podobná závislost se projevila také u Apache Storm, který ve třech scénářích podával relativně konstantní výkonnost, ale při použití konfigurace *vm_medium* dokázal data zpracovat přibližně o 50 % rychleji.



Obrázek 5.8: Srovnání výkonu na konfiguraci 4x *vm_small*.



Obrázek 5.9: Srovnání výkonu na konfiguraci 4x *vm_medium*.

Kapitola 6

Detekce útoků na autentizaci RDP

Cílem praktické části mé práce bylo kromě implementace výkonnostních testů také naprogramování vybrané metody pro detekci síťových útoků pomocí proudového zpracování dat.

Pro tyto účely jsem zvolil detekci útoků na autentizaci protokolu RDP (Remote Desktop Protocol). Tento protokol slouží pro připojení ke vzdálené ploše strojů s operačním systémem Microsoft Windows a zjednodušuje tak vzdálenou správu počítačů a serverů. Poprvé byl představen ve Windows NT 4.0 Terminal Server Edition a od verze Windows XP obsahují operační systémy Windows předinstalované služby Remote Desktop Connection a Remote Desktop Services, díky kterým se lze na vzdálenou plochu připojit.

Pro úspěšné připojení je nutné se autentizovat přihlašovacím jménem a heslem, což činí protokol RDP častým cílem útoků hádajících hesla. V současné době existuje celá řada automatizovaných nástrojů pro takové útoky a pro zajištění bezpečnosti sítě je nutné je odhalit a mitigovat dříve, než dojde ke kompromitaci systému.

Detailní analýzu odhalování útoků na autentizaci RDP pomocí technologie NetFlow provedl Martin Vizváry ve své diplomové práci [27], kde identifikoval nejčastější vzory síťové komunikace probíhajícího útoku. Díky tomu jsem se mohl zaměřit na implementaci detekce v proudovém paradigmatu s využitím těchto vzorů. Po analýze nejčastěji používaných RDP klientů a nástrojů pro útoky byly stanoveny parametry síťového toku probíhajícího útoku směrem od útočníka k oběti následovně:

- cílový port 3389
- TCP příznaky ACK, PUSH, SYN a FIN
- počet odeslaných paketů v intervalu 20 až 100
- počet přenesených bajtů v intervalu 2200 až 8001

Pro komunikaci opačným směrem od oběti k útočníkovi jsou parametry velmi podobné:

- zdrojový port 3389
- TCP příznaky ACK, PUSH, RST a SYN
- počet odeslaných paketů v intervalu 30 až 190
- počet přenesených bajtů v intervalu 3000 až 180000

Pro zpřesnění detekce a odstranění případných false positive detekcí je navíc pokusům o přihlášení předřazena podmínka, že útočník musí nejprve provést sken sítě na portu 3389 v definovaném časovém úseku před samotným útokem, čímž se vyloučí detekce legitimních připojení.

Všechny výše uvedené parametry komunikace jsem implementoval pomocí konfiguračního souboru a pro jejich změnu v instanci detekční metody tak stačí pouze přepsat hodnoty v textovém souboru a restartovat příslušný *Samza job*.

6.1 Časové okno

Základem detekce útoků na autentizaci je nastavení limitu počtu pokusů o přihlášení za určitý časový úsek. To v kontextu proudového zpracování dat znamená nutnost uchovávat data za celý úsek a analýzu dělat nad nimi.

V mé implementaci jsem koncept časových oken pojal mírně netradičně a rozhodl se ukládat do paměti pouze toky na portu 3389 se správně nastavenými TCP flagy, což má za následek výrazné snížení paměťové náročnosti a zrychlení výpočtů, protože tak odpadá neustálé zpracovávání dat nesouvisejících s útoky.

Pro tvorbu časového okna jsou použity časové známky přímo z příchozích toků a analýza je tak nezávislá na synchronizaci jednotlivých strojů (v rámci výpočetního clusteru) i na pořadí příchodu toků. Za předpokladu, že komunikace mezi útočníkem a obětí je směrována stejnou linkou, navíc odpadá potřeba synchonizace jednotlivých síťových sond.

Posledním důležitým parametrem časových oken je způsob a rychlost aktualizace. Díky využití časových známek toků jsem mohl implementovat aktualizaci čistě v reálném čase namísto pevně zvoleného intervalu. S příchodem každého toku je nejprve rozhodnuto, zda se bude ukládat, poté následuje extrakce zdrojové IP adresy a časové známky. Pro tuto adresu jsou pak vyhledány uložené toky a smazány všechny takové, které jsou starší než aktuální časová známka mínus velikost okna (definovaná v konfiguraci). Tento mechanismus zaručuje, že pro detekci budou k dispozici vždy ta nejnovější možná data a zároveň snižuje výpočetní nároky na udržování okna, kdy jsou zpracovány pouze toky pro konkrétní IP adresu.

6.2 Detekce útoku

Samotná detekce útoku je v prostředí Apache Samza přímočará. Z důvodu vysokých nároků, které Samza klade na tvorbu vstupního proudu a rozdělení dat do jeho oddílů, není potřeba řešit sdílení dat mezi výpočetními uzly. Tím je zajištěno, že každá úloha dostane veškerá potřebná data a celý proces detekce tak zvládne jediný Samza *job*.

Detekce útoku poté probíhá v návaznosti na aktualizaci časového okna. Po vyčištění starých toků jsou všechny relevantní záznamy zpracovány a je spočítán počet toků odpovídající výše popsaným kritériím. Pokud tento počet překročí stanovenou hranici v obou směrech komunikace, je do Kafka tématu odeslána zpráva o detekovaném útoku obsahující IP adresu útočníka, počet detekovaných pokusů o přihlášení a časová známka (do té doby) posledního toku útoku.

Vzhledem k povaze proudového zpracování by takto docházelo k reportování útoku s každým příchozím tokem, a proto jsem do detekce implementoval možnost nakonfigurovat frekvenci hlášení pomocí parametru udávajícího čas od posledního hlášení, kdy je možné odeslat novou zprávu o útoku.

Zbývající komponentou detekce útoků je detekce skenování sítě. Tento předpoklad je definován jako příjem toku s cílovým portem 3389 a TCP příznakem SYN, přičemž ostatní příznaky jsou nastaveny na nulu. Skenování je následně přiřazeno zdrojové IP adrese a uloženo spolu s časovou známkou, aby bylo možné během analýzy vyloučit detekování připojení bez předchozího skenu, nebo s příliš starým skenem. Podmínka předcházejícího skenu je ovšem velmi restriktivní a detekční nástroje ji obvykle ignorují s cílem pokrýt větší množství útoků. Z tohoto důvodu jsem do konfigurace metody přidal možnost nebrat sken jako nutnou podmínku.

6.3 Testování a porovnání s Flowmon ADS

Pro otestování detekce útoků jsem použil reálná data z provozu Masarykovy univerzity. Ta jsem konvertoval do JSON formátu popsaného v kapitole o výkonnostním testování a spustil nad nimi implementovanou metodu detekce útoků.

Účelem testování nebylo měření výkonu, ale ověření správnosti detekce, a probíhalo tak v prostředí Oracle VM VirtualBox na mém notebooku. Pro běh operačního systému Ubuntu 16.04 bylo vyčleněno jedno jádro procesoru Intel Core i7-5500U a 4GB operační paměti. Na této konfiguraci byly spuštěny systémy Apache Kafka pro doručování zpráv, Ha-

doop YARN pro správu clusteru a Apache Samza.

Odesílání dat pro detekci probíhalo pomocí jednoduchého skriptu *kafka-console-producer*, který je běžnou součástí distribuce Apache Kafka a zprávy zvládal odesílat rychlostí okolo 100k toků za vteřinu. Tento objem dat moje detekční metoda dokázala i v takto výrazně omezených podmínkách zpracovat.

Výhodou použití reálných dat je možnost přímo porovnat výsledky detekce proti produkčně nasazenému systému Flowmon ADS 8.00 obsahující metodu detekce RDP útoků popsanou v teoretické části práce. Vzhledem k tomu, že tato metoda vycházela v rámci smluvního výzkumu se společností Flowmon Networks ze stejné analýzy reálných útoků jako moje implementace, je možné výstupy obou systému porovnávat. Pro to je ovšem potřeba v obou metodách nakonfigurovat shodné parametry, především velikost časového okna a limit počtu přihlášení. Během testování byly tyto parametry nastaveny na hodnotu 15 pokusů a přihlášení za posledních 10 minut.

Z útoků zachycených v provozu sítě univerzity dokázala metoda v Apache Samza detekovat téměř všechny známé útoky. Jedinou skupinou nedetekovaných útoků byly takové, jejichž toky neobsahovaly příznak reset (RST) nebo finish (FIN), což byl zároveň jediný rozdíl konfigurací proudové metody a ADS. Detekce reálných útoků tak potvrdila schopnosti proudového zpracování dat odhalit stejné útoky jako při použití dávkové varianty.

Závěrečným experimentem prováděným v rámci mé práce bylo srovnání včasnosti detekce proudového paradigmatu vůči dávkovému zpracování. V klasickém pojetí probíhá analýza jednou za 5 minut a v detekci tak vzniká prodleva mezi překročením limitu a hlášením události. Naopak proudové zpracování dat provádí analýzu kontinuálně a reportování události by tak mělo proběhnout okamžitě po jejím detekování.

Dalo by se předpokládat, že útoky budou v daném pětiminutovém intervalu rozložené rovnoměrně a zpoždění detekce tak bude v průměru dvě a půl minuty. Ověření této teorie shrnuje tabulka 6.1, ve které jsou zaznamenány útoky proti univerzitě zachycené pomocí Flowmon ADS. U každého útoku byl zaznamenán čas detekce a příslušné NetFlow záznamy síťové aktivity. Ty byly dále přeposlány do detekční metody v Apache Samza a znovu detekovány, třetí sloupec tabulky pak obsahuje rozdíl těchto dvou hodnot. Nutno podotknout, že Flowmon ADS uvádí jako čas detekce začátek intervalu, ve kterém byla spuštěna analýza, což zakrývá zpoždění samotné analýzy dat celého intervalu. Podobný problém je i v mé metodě, která jako čas detekce uvádí časovou známku konce

toku a neobsahuje tak dobu potřebnou pro export do systému a zpracování (latence sítě a zpracování v Apache Samza se pohybuje v řádech zlomků vteřiny).

Z naměřených dat vyplývá, že průměrné zrychlení detekce v proudovém paradigmatu činí 181,79s. Tato odchylka od očekávaného výsledku je způsobena útoky, které začnou na konci detekčního intervalu a limit pokusů o přihlášení překročí až na začátku druhého intervalu, což způsobí vychýlení průměru k vyšším hodnotám.

Čas detekce v ADS	**Čas detekce v Apache Samza**	**Rozdíl [s]**
2016-04-27 10:40:00	2016-04-27 10:38:15	105
2016-04-27 10:45:00	2016-04-27 10:40:21	279
2016-04-27 10:25:00	2016-04-27 10:24:11	49
2016-04-27 10:25:00	2016-04-27 10:22:21	159
2016-04-27 10:25:00	2016-04-27 10:22:37	143
2016-04-27 10:25:00	2016-04-27 10:21:33	207
2016-04-27 09:50:00	2016-04-27 09:46:17	223
2016-04-27 09:40:00	2016-04-27 09:37:31	149
2016-04-27 09:35:00	2016-04-27 09:33:35	85
2016-04-27 09:25:00	2016-04-27 09:22:35	145
2016-04-27 09:15:00	2016-04-27 09:13:23	97
2016-04-27 09:10:00	2016-04-27 09:05:34	266
2016-04-27 08:40:00	2016-04-27 08:37:18	162
2016-04-27 08:20:00	2016-04-27 08:15:52	248
2016-04-27 08:10:00	2016-04-27 08:07:15	165
2016-04-30 07:15:00	2016-04-30 07:11:09	231
2016-04-30 06:20:00	2016-04-30 06:16:44	196
2016-04-30 06:15:00	2016-04-30 06:12:03	177
2016-04-30 06:05:00	2016-04-30 06:01:38	202
2016-04-30 05:50:00	2016-04-30 05:47:35	145
2016-04-30 05:40:00	2016-04-30 05:36:42	198
2016-04-30 05:15:00	2016-04-30 05:10:13	287
2016-04-30 04:40:00	2016-04-30 04:35:06	294
2016-04-30 04:40:00	2016-04-30 04:37:02	178
2016-04-30 04:20:00	2016-04-30 04:18:25	95
2016-04-30 03:50:00	2016-04-30 03:46:28	212
2016-04-30 03:40:00	2016-04-30 03:37:07	173
2016-04-30 03:10:00	2016-04-30 03:07:01	179
2016-04-30 02:25:00	2016-04-30 02:21:17	223
Průměr	–	181,79

Tabulka 6.1: Rozdíly v časech detekce různých paradigmat zpracování dat

Kapitola 7

Závěr

Ve své diplomové práci jsem se zabýval využitím paradigmatu proudového zpracování dat pro účely bezpečnostní analýzy síťového provozu. V teoretické části práce jsem popsal state-of-the-art v oblasti monitorování síťové aktivity pomocí technologie NetFlow se zaměřením na problematiku detekce síťových útoků. Současné systémy detekcí postavené nad flow daty však pracují klasickým dávkovým způsobem a proudové zpracování dat popsané ve třetí kapitole tak představuje možnost reálného zrychlení detekce útoků.

Následující kapitola se věnuje třem v současnosti nejpoužívanějším systémům pro proudové zpracování dat – Apache Samza, Apache Spark a Apache Storm. Detailně je zde představen systém Apache Samza a jeho nástroje pro správu výpočetního clusteru a distribuované výpočty nad datovým modelem příjmu a odesílání zpráv. U zbylých systémů jsou pak diskutovány jejich odlišnosti od Samzy a na závěr je uvedeno souhrnné srovnání všech tří frameworků.

Praktická část práce je věnována vývoji pro Apache Samza. Nejprve jsem implementoval testovací metody pro výkonnostní srovnání frameworků proudového zpracování dat. Tento vývoj probíhal v rámci výzkumného projektu *TA04010062/2014 Technologie pro zpracování a analýzu síťových dat velkého rozsahu* sponzorovaného Technologickou agenturou ČR v projektovém rámci ALFA – Program na podporu aplikovaného výzkumu a experimentálního vývoje. Výsledky tohoto testování byly dále zpracovány a publikovány v odborném článku *A Performance Benchmark for NetFlow Data Analysis on Distributed Stream Processing Systems* [13]. Zdrojové kódy výkonnostního testování a nástroje pro převod datasetu [7] do JSON formátu jsou k dispozici ke stažení ve veřejném univerzitním repozitáři[1].

Pokračováním praktické části práce byla implementace detekční metody schopné odhalit síťový útok pomocí proudového zpracování dat, jejímž cílem je demonstrovat možnost využití tohoto paradigmatu pro

1. https://is.muni.cz/repo/1323006/dsp-systems-benchmark.zip

bezpečnostní analýzu. K tomu účelu jsem zvolil detekci útoků proti autentizaci služby vzdálené plochy v systémech Microsoft Windows.

V závěru praktické části jsou pak experimentálně ověřeny přínosy proudového zpracování dat z hlediska včasné detekce útoků, kdy jsou porovnány detekční časy zástupce dávkového zpracování (systém Flowmon ADS 8.00) proti mnou implementované metodě. Výsledky tohoto testu ukazují, že se stejnou konfigurací detekce přináší nové paradigma zrychlení odhalení útoku v řádech minut, což může významně ovlivnit úspěšnost útočníka proniknout do systému. Zde musím zmínit skutečnost, že krátce před odevzdáním této diplomové práce vydala společnost Flowmon Networks novou verzi systému ADS, 8.01.00, ve které zavádí prvky proudového zpracování a nabízí profily s granularitou 30 vteřin výrazně urychlující detekci útoků. Tím dokazuje, že využití tohoto paradigmatu má smysl i v reálném komerčním prostředí.

Ve výzkumu oblasti proudového zpracování dat vidím obrovský potenciál i do budoucna a to nejen v oblasti bezpečnosti, ale i samotného monitorování síťového provozu a správy sítí. Možnou cestu uplatnění v praxi ukazuje článek *Real-time Analysis of NetFlow Data for Generating Network Traffic Statistics using Apache Spark* [12], jehož jsem spoluautorem a demonstruje využití proudového zpracování pro tvorbu statistik provozu.

Literatura

[1] Apache Software Foundation. Apache Storm, 2014. [cit. 6. 8. 2015]. Dostupné z: https://storm.apache.org/.

[2] Apache Software Foundation. Apache Hadoop NextGen MapReduce (YARN), 2015. [cit. 9. 8. 2015]. Dostupné z: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[3] Apache Software Foundation. Samza, 2015. [cit. 6. 8. 2015]. Dostupné z: http://samza.apache.org/.

[4] Apache Software Foundation. Spark Streaming Programming Guide, 2015. [cit. 28. 3. 2016]. Dostupné z: http://spark.apache.org/docs/latest/streaming-programming-guide.html.

[5] Apache Software Foundation. Welcome to Apache Hadoop, 2016. [cit. 7. 3. 2016]. Dostupné z: http://hadoop.apache.org/.

[6] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[7] CAIDA. The CAIDA UCSD Anonymized Internet Traces 2015 - 20150219-130000, 2015. [cit. 2. 9. 2015]. Dostupné z: http://www.caida.org/data/passive/passive_2015_dataset.xml.

[8] Inc. Cisco Systems. NetFlow Services Solutions Guide, 2001. [cit. 26. 2. 2016]. Dostupné z: http://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/netflow/nfwhite.html.

[9] Inc. Cisco Systems. NetFlow Export Datagram Formats, 2007. [cit. 16. 2. 2016]. Dostupné z: http://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.pdf.

[10] Inc. Cisco Systems. Introduction to Cisco IOS NetFlow - A Technical Overview, 2012. [cit. 17. 2. 2016]. Dostupné z: http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.pdf.

[11] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), 2008.

[12] Milan Čermák, Tomáš Jirsík, and Martin Laštovička. Real-time analysis of netflow data for generating network traffic statistics using apache spark. In *IEEE/IFIP Network Operations and Management Symposium 2016 (NOMS 2016)*, pages 1019–1020, Istanbul, Turkey, 2016. IEEE Xplore Digital Library.

[13] Milan Čermák, Daniel Tovarňák, Martin Laštovička, and Pavel Čeleda. A performance benchmark of netflow data analysis on distributed stream processing systems. In *IEEE/IFIP Network Operations and Management Symposium 2016 (NOMS 2016)*, pages 919–924, Istanbul, Turkey, 2016. IEEE Xplore Digital Library.

[14] Jinliang Fan, Jun Xu, and Mostafa Ammar. Cryptography-based Prefix-preserving Anonymization, 2006. [cit. 23. 5. 2016]. Dostupné z: http://www.cc.gatech.edu/computing/Telecomm/projects/cryptopan/.

[15] A.S. Flowmon Networks. Flowmon ADS Enterprise 7.02.00 Uživatelská příručka, 2015.

[16] A.S. Flowmon Networks. Detekce anomálií & analýza chování sítě, 2016. [cit. 26. 2. 2016]. Dostupné z: https://www.flowmon.com/cs/solutions/use-case/network-behavior-analysis-anomaly-detection.

[17] A.S. Flowmon Networks. Netflow, nová éra monitorování počitačových sítí, 2016. [cit. 26. 3. 2016]. Dostupné z: https://www.flowmon.com/cs/solutions/use-case/netflow-ipfix.

[18] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, June 2003.

[19] Peter Haag. NFDUMP. Web page, December 2014. Accessed August 6, 2015.

[20] Peter Haag. NfSen. Web page, 2015. Accessed August 26, 2015.

[21] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: A distributed messaging system for log processing. In *Proceedings of 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece*, 2011.

[22] Martin Laštovička. Dolování ip charakteristik z netflow. Bakalářská práce, Masarykova univerzita, 2014.

[23] Ruirui Lu, Gang Wu, Bin Xie, and Jingtong Hu. Streambench: Towards benchmarking modern distributed stream computing frameworks. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, pages 69–78, December 2014.

[24] T. Porter. The perils of deep packet inspection. *Security Focus*, 2005. Dostupné z: http://www.symantec.com/connect/articles/perils-deep-packet-inspection.

[25] Ramesh, N. Apache Samza, LinkedIn's Framework for Stream Processing, 2015. [cit. 19. 3. 2016]. Dostupné z: http://thenewstack.io/apache-samza-linkedins-framework-for-stream-processing.

[26] Cisco Systems. Cisco ios netflow, 2014. [cit. 22. 1. 2014]. Dostupné z: http://www.cisco.com/go/netflow.

[27] Martin Vizváry. Detekcia útokov na autentizáciu rdp. Diplomová práce, Masarykova univerzita, 2012.

[28] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10. USENIX Association, 2010.

Obsah přiloženého CD

- Text práce ve formátu PDF.
- Zdrojové kódy výkonnostního testování.
- Zdrojové kódy detekční metody.