ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Podaný** Jméno: **Pavel** Osobní číslo: **397850**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Vizualizace síťových toků v infrastruktuře cloudu

Název diplomové práce anglicky:

Visualization of network flow in cloud infrastructure

Pokyny pro vypracování:

Analyzujte techniky pro vizualizaci toků v sítích a jejich aplikace na toky v počítačových sítích (např. toky mezi servery cloudu). Na základě analýzy navrhněte a implementujte aplikaci schopnou vizualizovat skutečné toky dat mezi virtuálními servery cloudu. Zaměřte se na vizualizaci a analýzu toků v jednom daném časovém okamžiku. Dále se zaměřte na vizualizaci relací mezi servery a virtuálními servery. Výslednou aplikaci otestujte na pěti datových sadách různé složitosti získaných z reálného cloudu (či jeho simulace) sestávajícího alespoň z deseti serverů a třiceti virtuálních serverů. Cílem testování je vyhodnotit časovou a paměťovou náročnost implementované aplikace v závislosti na složitosti dat a demonstrovat škálovatelnost navržené vizualizace (pro jak složitá data je vizualizace použitelná).

Seznam doporučené literatury:

[1] Tamassia, R. (Ed.). Handbook of graph drawing and visualization. CRC Press, 2013.
[2] Shiravi, H., Shiravi, A. and Ghorbani, A. A. A survey of visualization systems for network security. IEEE Transactions on visualization and computer graphics, 18(8), 1313-1329, 2012.
[3] Braun, L., Volke, M., Schlamp, J., von Bodisco, A. and Carle, G. Flow-inspector: a framework for visualizing network flow data using current web technologies. Computing, 96(1), 15-26, 2014.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Ladislav Čmolík, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **14.02.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

Ing. Ladislav Čmolík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Diplomová práce

Vizualizace síťových toků v infrastruktuře cloudu

Bc. Pavel Podaný

Vedoucí práce: Ing. Ladislav Čmolík, Ph.D

23. května 2018

Poděkování

Rád bych poděkoval vedoucímu této práce, panu Ing. Ladislavu Čmolíkovi, Ph.D, za jeho rady, postřehy a četné informace, bez kterých by tato práce nebyla možná.
Také děkuji panu Ing. Jiřímu Chludilovi za čas věnovaný konzultacím k systému BigCloud, které byly během tvorby zapotřebí.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 23. května 2018 .....

Odkaz na tuto práci

Podaný, Pavel. *Vizualizace síťových toků v infrastruktuře cloudu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2018.

Abstrakt

Cloudové počítačové systémy v moderním světě dosahující stále větší presence a komplexnosti a mnohé z nich obsahují stovky či tisíce unikátních vnitřních sítí. Kvalitní monitoring takovéto infrastruktury poté může být extrémně složitý nebo i nemožný.
Tato práce se zaobírá rozborem reálného cloudového systému BigCloud zaměřeného na tvorbu a správu virtuálních strojů a sítí mezi nimi a analýzou, návrhem a implementací webové aplikace, která by dokázala vizualizovat datové toky uvnitř virtuálních sítí (nejen) tohoto cloudu.

Klíčová slova vizualizace, počítačové sítě, datové toky, cloud, webová aplikace, d3.js, Symfony, kruhový diagram, heatmapa

Abstract

Modern cloud computer systems show an increasing growth in presence and complexity and as such many of them can contain hundreds or even thousands of unique inner networks. Such infrastructures might then become very hard or even impossible to monitor in any meaningful way.
This work takes a look at one such cloud system, Bigcloud, focused on creating and management of virtual machines and the networks that connect them,

and attempts to analyze, design and implement a web application capable of visualizing data flows of the networks inside of (not only) this cloud.

Keywords visualization, computer networks, data flows, cloud, web application, d3.js, Symfony, circular diagram, heatmap

Obsah

Úvod

Téma vizualizace dat a dalších informací je s lidstvem mnohá staletí, od jednoduchých plánů a náčrtků až po zobrazení komplexních měření s automatickou analýzou, lidé za svou existenci zobrazili nepřeberné množství informací. Vizualizace sítí a datových toků je v porovnání se svou mateřskou disciplínou velmi mladým oborem, který ve svých počátcích nebyl ani příliš potřebný, vzhledem k jednoduchosti komunikačních architektur a malému množství dat v nich protékajících.
Rozsáhlost informačních technologií však masivním tempem roste a poptávka po rychlejších a komplexnějších sítích spolu s nimi. Velikost i lokálních pokrytí s množstvím hardwarových či softwarových zařízení, které tok dat upravují tedy stále častěji vede k otázce co se vlastně v síti děje.
První z protokolů pro sledování toků dat, Netflow, byl vyvinut roku 1996 a patentován firmou Cisco. Ranné studie a průzkumu o využití dat touto formou získaných následovaly brzy na to. Dá se tedy říci, že se jedná o pouhá dvě desetiletí starý obor. Ten si přesto, obzvláště v posledních letech, začíná získávat mnohé pozornosti, například i kvůli stále častějšímu výskytu tzv. cloudů.

Cloud

Pojem, jehož využití ve světě informatiky se poslední dobou objevuje stále více. Co to ale vlastně je?
Existuje mnoho specifických služeb a architektur takto provozovaných, zhruba se však jedná o variabilní množství výpočetních prostředků, které jsou, většinou za příslušné poplatky, poskytovány uživatelům k využití nejrůznějších služeb. Tyto mohou být širokého rozsahu, od infrastruktury pro ukládání dat firemní aplikace po poskytování rozhraní k vytváření vlastních virtuálních serverů dle požadovaných konfigurací. Podstatnou částí však je, že se uživatel nestará o hardwarové specifikace strojů tyto služby poskytující, rozdělení systémových a fyzických prostředků, pokud jsou tyto dostupné, je prováděno au-

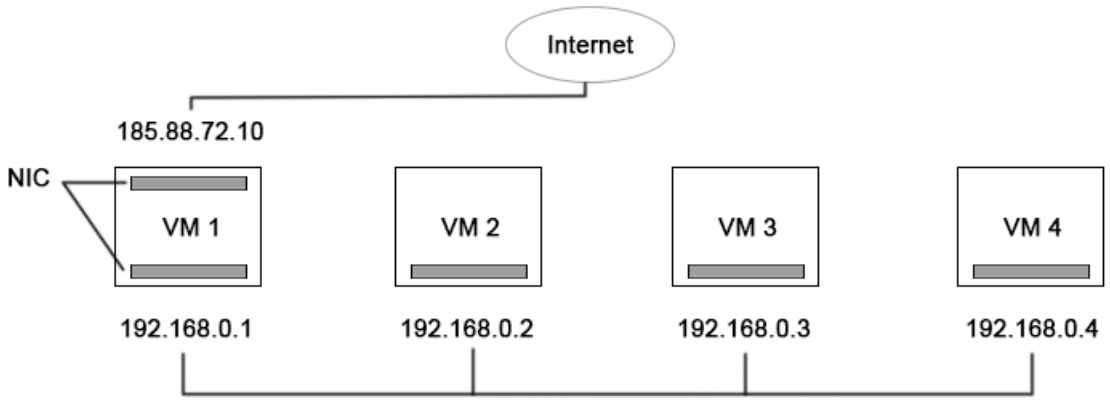
tomaticky příslušnou infrastrukturou. Základní funkčnost cloudu přitom není omezena tím, zda-li se k dispozici nachází jediný server nebo několik tisíc, tyto lze do systému libovolně přidávat a odebírat dle potřeb provozovatele a tím vytvářet komplexní nasazení s obrovským výpočetním potenciálem, které většinou nevyžaduje ke svému použití (ačkoli ne nezbytně konfiguraci) žádné speciální schopnosti.
Provozované servery se mohou vyskytovat ve stejné geografické lokaci, je však také možné, aby byly rozprostřeny po celé planetě v různých částech kontinentů. Nehledě na rozsáhlost implementace, jednotlivé zdroje jsou vždy propojeny jednou nebo více sítěmi (ty mohou například být rozdělené na provozní, administrativní, nebo virtuálně rozkouskované), které mohou velmi brzy dosáhnout velice komplikovaných struktur. Je tedy žádané mít k dispozici nástroj schopný zobrazit aktuální stav a rozsah těchto prostředků.

Specifikace monitorované infrastruktury

Jak již bylo zmíněno v předchozí části, forem cloudů je mnoho a je tedy nutné ustanovit detaily toho, kterým se tato práce bude zabývat.
Projekt BigCloud, na který se tento text zaměřuje, představuje ideu infrastruktury cílené na jednoduché vytváření a správu virtuálních strojů (Virtual Machine, dále VM) za pomocí webového rozhraní. Uživateli se tímto způsobem nabízí možnosti vytvoření libovolně rozsáhlé a výpočetně/paměťově schopné (v rámci možností použitých prostředků) sítě virtuálních serverů k použití pro vlastní účely. Tímto způsobem si například lze zřídit server k nainstalování webového serveru a po přiřazení veřejné IP adresy na tento stroj (opět přes příslušné webové rozhraní) ho k těmto účelům používat.
Architektura se tedy mimo jiné stará o alokaci zdrojů, vytvoření samotného VM, jeho správu (zapnutí, vypnutí, restart, editace konfigurace virtuálního hardwaru) a v neposlední řadě o vytvoření virtuálních sítí na kterých se tyto stroje budou nacházet.
Jelikož je pro provoz infrastruktury použito technologie virtuální sítě (VLAN), lze na jedné fyzické síti provozovat velké množství izolovaných podsítí a tím předejít jakýmkoli nežádoucím komunikacím mezi stroji různých uživatelů (tyto spolu po vnitřní síti jednoduše nedokážou komunikovat). Z toho tvrzení vyplývá, že celá (přinejmenším provozní část, administrativní a jiné součásti v tomto textu neuvažujeme, jelikož nejsou součástí cílů vizualizace) síťová struktura BigCloudu se skládá z libovolného (a tedy i velkého) množství izolovaných podsítí, jejichž rozsah sám o sobě s největší pravděpodobností nebude přesahovat množství několika jednotek VM.
Tento předpoklad infrastruktury bude později hrát klíčovou roli pro cíle práce.

Jednou z dalších příjemných vlastností BigCloudu je to, že je psán komplet od základů firmou, která ho vyvíjí. Na trhu lze najít množství jak proprietárních

tak opensource řešení, s jejichž pomocí lze jednoduše zřídit podobné služby, mezi jejich nevýhody však mimo jiné patří obtížná či nemožná modifikovatelnost a přístup do základních vrstev funkčnosti.
Toto poskytuje možnosti jako je přímá práce se získáváním dat, znalosti použitých prostředků a technologií a dalších benefitů pro napsání vlastního vizualizačního nástroje.



Obrázek 0.1: Příklad zapojení sítě virtuálních strojů

Obrázek 0.1 nám ukazuje příklad zapojení jedné virtuální sítě. Jednotlivé virtuální stroje disponují libovolným počtem NIC (Network Interface Controller - síťová karta). V tomto příkladu mají VM 2 až 4 po jednom rozhraní, která jsou použita pro připojení k privátní síti (192.168.0.1 - 192.168.0.4). VM 1 má přiřazeno o jedno rozhraní navíc, které disponuje veřejnou IP adresou. Tato adresa stroji dovoluje přístup do internetu.
V případě správného nastavení směrování je možné aby i ostatní stroje přes tuto adresu přistupovali ven, pokud budou VM 1 používat jako bránu do internetu. Tato možnost však v některých případech může být limitována bezpečnostními protokoly cloudu.

Cíle práce

Cílem textu a následné implementace je návrh a realizace nástroje pro snadný přehled struktury a stavu zatížení sítě výše popsaného cloudu, Bigcloud a všech jeho podsítí.
Ačkoli bude nástroj svou konstrukcí zaměřen na tuto infrastrukturu, mělo by být možné jej využít i pro zobrazení jiných nezávislých cloudů podobné struktury, pokud tyto budou schopny poskytnout nezbytná data v požadovaném formátu.

Nejprve si vypíšeme typy síťových vizualizací, pro jaké účely se používají a pokusíme se vybrat ten typ k naším cílům nejbližší. V této části bude nutné blíže specifikovat požadavky nástroje a co přesně si od jeho funkčnosti představujeme.
V další části se zaměříme na popis obecných technik pro vizualizaci toků a analýzu již existujících řešení síťových vizualizací, které se na trhu vyskytují, jejich použití, klady a zápory a vhodnost aplikovaných technik pro použití v naší aplikaci. Tato analýza bude zakončena shrnutím nabitých znalostí a případných modifikací, které budou nezbytné.
Tyto průzkumy by nám měly být schopné poskytnout dostatečné množství znalostí k návrhu co možná nejlepší vizualizační metodiky pro zamýšlené použití.
Také si specifikujeme nezbytnou architekturu cílového nástroje, sběrače dat, jejich modulů a dalších nezbytných součástí výsledného produktu. Posledním krokem analýzy bude průzkum dostupných technologií, existujících protokolů využívaných v podobných nástrojích, programovacích jazyků a volba implementačních technologií naší aplikace.
Po dokončení analýzy a návrhu se v práci blíže podíváme na jednotlivé kroky implementace a jak bylo co provedeno a nakonec výsledný produkt otestujeme na množství vygenerovaných datasetů a spolu s tím prozkoumáme škálovatelnost naší implementace.

Kapitola 1

Možné druhy vizualizace sítě

Stejně jako v množství jiných oborů zaobývajících se zobrazením dat, klíčovým aspektem při návrhu takovéto aplikace je znalost toho, co vlastně chceme vidět. Tato vlastnost je tak trochu paradox celé vizualizace, jelikož ta se často provádí proto, abychom tento fakt vůbec mohli zjistit (v datech se snažíme vidět nějaké zajímavé chování). Nicméně pro inspiraci se můžeme podívat na to, jaké případy síťové vizualizace se v praxi používají.

1.1 Výčet možností

Začněme výčtem podstatných odvětví, pro které mohou být vyžadovány odlišné přístupy k problému. V další části si poté ke každému z těchto bodů řekneme něco více.

- Hierarchie sítě a jejích prvků
- Množství dat, které si jedna IP adresa vymění s ostatními
  - Na lokální síti
  - Na internetu
  - Na lokální síti a/nebo internetu
- Připojitelnost prvků sítě
- Detekce útoků či abnormalit
- Bezpečnost sítě
- Profilování uživatelů na základě dat

1.2 Detailnější pohled

Některé z těchto odvětví se vydávají zcela novým směrem, zatímco jiné mohou s těmi ostatními sdílet množství technik a předpokladů, podívejme se tedy blíže na to, k čemu se tyto metodiky využívají.

- **Hierarchie sítě a jejích prvků**
  Pro sítě většího rozsahu než domácího může být složité zachovávat přehledný plán jejich rozsahu a pozice jednotlivých elementů. Tento typ vizualizace slouží k přehlednému namapování všech prvků sítě, jejich propojení a případně dalších užitečných informací (například poruchy spojení mezi stroji). Toho může být dosaženo jak využitím geografických dat pro konstrukci realisticky přesného plánu sítě, nebo více abstraktní metodou jako například silou rozprostřeným grafem. Také lze volit mezi vykreslením ve 2D ve formě plánků nebo komplexnější 3D formou pro využití například ve virtuální realitě.
  Kladnými vlastnostmi jsou zde především přehlednost a jednoduchost, jelikož se jedná o pouhé binární relace spojů (existuje/neexistuje).
  Mezi záporné body patří nedostatečná informativnost k úkolům více komplexním než-li nalezení lokace specifického prvku nebo informací o návaznosti strojů.

- **Tok dat v lokální sítí**
  Uživatele častokrát může zajímat množství jakým způsobem jsou linky mezi hosty využívány, to znamená který stroj posílá jaké množství dat kterému, přes které uzly tato data musejí projít, co se s nimi v průběhu cesty děje a případně jak se tato chování mění v závislosti na časovém okamžiku (ať již se jedná o diference časových intervalů nebo zobrazení dat pro zadaný časový úsek). Tento úkol se potýká zejména s problémy přehledného zobrazení všech existujících spojení a velikosti dat, které přes ně protékají. Nastávají zde otázky vztahu zaznamenaných dat k relevanci jejich velikosti (zobrazovat velikosti linků podle min/max hranic zobrazovaných dat, nějakých praxí stanovených hranic nebo za pomocí úplně jiných parametrů) a přehlednosti zobrazovaných diagramů pro velké množství dat.
  Výhody jsou bezesporu schopnost okamžitého rozpoznání velice aktivních úseků sítě, které mohou vyžadovat bližší pozornost případně naopak nevyužitých linek, pro které by se mohlo najít lepší využití.
  Problémem může být absence možnosti snadné identifikace návaznosti sítí a podsítí a tím rychlá identifikace fyzické části zajímavého úseku. Jedná se zde zejména o vztahy typu N:M, lze tedy předpokládat velké množství vykreslených dat, které se často mohou nepříjemně křížit.

- **Tok dat do internetu**
  Podobně jako v předchozím případě se zde zajímáme o toky a velikosti

dat mezi mnoha stroji, zde se však bude jedna podstatně více o vizualizace typu 1:N, kde sledujeme komunikaci jednoho stroje (či IP adresy) v naší síti s mnoha stroji nacházejících se v internetu.
Tento způsob zobrazení nabízí snadné identifikace externích zařízení, která linku či stroj nejvíce vytěžují (například zasílají velké množství dat), je však nutné počítat s možností obrovského množství různých adres, které se na stroj připojují, což se může lehce stát například v případě webového serveru. V závislosti na tom, k čemu nám daná vizualizace slouží tento problém lze řešit například omezením zobrazovaných adres na určitý počet těch nejzajímavějších (největší datová zátěž, nejvyšší počet připojení atd.).

- **Tok dat v lokální síti a internetu**
  Kombinace obou předchozích variant zredukovaná do jediného diagramu. Tento případ je velmi závislý na kontextu zkoumaného problému, jednoduše se totiž může stát, že přehledné zobrazení obou problémů naráz nebude dost dobře možné. To lze částečně řešit zvolením ústupků v detailnosti reportů každého z problémů a případně zkombinovat se samostatným zobrazením každého z obou případů pokud je u nějaké části požadováno vyšší detailnosti.

- **Připojitelnost prvků sítě**
  Mnohokrát se v praxi můžeme setkat s problémem nutnosti udržování maximální funkčnosti určitého množství spojení a případné co nejrychlejší reakce v případě výskytu selhání, například pokud nějaká množina serverů poskytuje vitální funkcionalitu a je tedy nutné se k nim být schopno v libovolnou dobu připojit. V tomto případě lze problém často vyřešit bez potřeby vstupovat do komplexních nákresů, které často vyžadují vysoké pozornosti nebo speciálního tréninku, ale lze využít jednoduchých alternativ jako kupříkladu matice aktivních spojení nebo list problémových součástí.
  Jako nevýhodu tohoto přístupu by šlo považovat přílišnou simplifikaci a jedno účelnost nástroje, do kterého v tomto stavu může být obtížné implementovat rozšiřující funkcionalitu.

- **Detekce útoků či abnormalit**
  Podstatně jiným problémem je případ elektronické bezpečnosti, kdy se snažíme zabránit či včasně identifikovat útoky hackerů, nebezpečné chování software a další potenciální bezpečností rizika počítačové sítě. Na rozdíl od předchozích případů zde probíraných, tento je již podstatně abstraktnější a zároveň také modernějším pohledem na vizualizaci sítí. Nástroje fungují na principu předvídavosti či znalosti chování počítačových virů, malwarů, způsobů neoprávněného vniknutí do sítě, krádeže dat a mnoho dalších způsobů kybernetických útoků. Cílem je zobrazovat data takovým způsobem, aby podezřelé vzorce chování byly patřičně

zvýrazněny, automaticky detekovány nebo alespoň takovouto schopnost uživateli poskytnout, je-li to nad technické možnosti aplikace.
Mezi možná využití patří zobrazení neobvyklé aktivity na portech, detekce podezřelých sekvencí paketů připomínající známý útok nebo nepředvídané akce a připojení. Toto umožňuje trénovanému uživateli včasně reagovat na případy, které, pokud by nedošlo k jejich včasné detekci a zastavení, by mohli způsobit značné finanční škody nebo únik citlivých informací.
Nevýhodou návrhu takového aplikace je nutnost předchozí znalosti vzorců chování počítačových útoků a schopnost aplikace učit se rozpoznávat stále nové elektronické hrozby.
Poskytované informace také často nebývají jednoduché ke čtení a uživatel tak může vyžadovat speciální trénink k získání těchto schopností.

- **Bezpečnost sítě**
  Tento případ vizualizace se zabývá schopností poskytování dat k zajištění požadované funkčnosti sítě. Může se jednat o zobrazení funkčnosti linek, jejich zatížení, redundantní spojení, možnosti přemostění vadného sektoru a mnoho dalšího. Dle specifických požadavků uživatele se může jednat o kombinaci přístupů v této sekci definovaných nebo nový na specifický účel zaměřený přístup.

- **Profilování uživatelů na základě dat**
  Rozpoznání uživatelů na základě určitých charakteristik, jako například otisku prstu, skenu sítnice nebo i stylu psaní na klávesnici není ve světě nic nového. Jako nový přírůstek do oboru se však ukazuje tato metodika, která využívá analýzu odeslaných a přijatých dat z uživatelova počítače k rozpoznání identity (například na základě dříve vytvořeného profilu), zkoumání jeho chování, detekce preferencí a mnoho dalších funkcionalit. Získaná data lze takto používat k velkému množství účelů, jako jsou bezpečnost, reklamní kampaně, sledování podezřelého chování a jiné.
  Jedná se o velice komplexní obor, který z velké části využívá lidské psychologie, předpokladů a neprozkoumaných oblastí.

1.3 Specifikace zacílení aplikace

Cílem naší práce je vytvoření nástroje umožňující zobrazení stavu sítě a toku dat v ní (množství přenesených dat mezi každými dvěma připojitelnými hosty), čímž se administrátorovi sítě za pomocí dalších poskytnutých nástrojů jako například filtrů umožní vyhledání problému. Tento záměr předpokládá jakési povědomí uživatele o možném problému v síti a umožňuje doladěním nastavení tuto zajímavou část izolovat a získat k ní specifické informace. Přestože hlavním cílem nástroje není okamžitá detekce jedné anomality, mělo by být možné jeho používáním spatřit podezřelé úseky a chování a tak získat zájem

o jejich bližší prozkoumání.
Již ze zadání vyplývá že se aplikace bude zabývat vizualizací toku dat a to především interních, uvnitř jednotlivých virtuálních sítí, bude však nabídnut i přehled zátěže na veřejných IP adresách s tokem do internetu. Tato část bude nicméně zkoumána v podstatně menším detailu, jelikož není možné (ani žádané) zobrazit přehled veškeré konektivity do internetu. Početní řád takovýchto připojení se na zaneprázdněných strojích může lehce pohybovat ve statisících či milionech a mimo velmi malé přínosnosti informace pro zde zkoumané účely bychom se také museli potýkat s exponenciálním nárůstem paměťové složitosti takto ukládaných souborů a tím také s podstatným navýšením datové zátěže při načítání informací pro vizualizaci.
Z předešlých případů se tedy nástroj nejvíce blíží případu využití pro vizualizaci toku dat v lokální síti s prvky vizualizace toku dat do internetu. Tyto informace lze však také využít pro další účely jako detekce útoků (abnormálně vysoký tok dat mnohokrát indikuje hackerský útok) nebo k selháním v sítí (například nulový tok dat mezi serverem a databází umístěných na odlišných strojích), v případě žádosti další specifické funkcionality ohledně této oblasti tedy lze z daných případů použití čerpat informace o možných filtrech a dalších nástrojích k využití.

1.4 Způsob hodnocení vizualizačních technik

Na základě této kapitoly jsme si blíže určili charakter aplikace, kterou se v této práce budeme snažit vytvořit. Pro tento účel se v dalších kapitolách budeme zabývat zkoumáním jak obecných, tak v jiných nástrojích použitých vizualizačních technik a vybrané kandidáty si bodově ohodnotíme k získání co možná nejlepší metodiky pro další práci.
Přestože ještě tyto techniky vybrány nemáme, způsob ohodnocení jejich schopnosti splnění našich požadavků bude vyplývat z této kapitoly a informací uvedených výše, je tedy vhodné si ho již nyní detailně specifikovat.
Vytvořme si tedy seznam požadovaných vlastností vizualizace oproti kterému budeme později nalezené vizualizační techniky hodnotit. Jelikož ne všechny z těchto vlastností jsou stejně kritické k zobrazení toho co potřebujeme, přiřadíme také každé z nich váhovou hodnotu ze stupnici od jedné do deseti (1 pro nejmenší relevanci, 10 pro kritickou), kterou bude poté hodnocení v dané kategorii násobeno.

- *Schopnost zobrazení toku dat mezi každou relevantní dvojicí adres v síti*
  Pravděpodobně nejpodstatnější potřebnou vlastností požadované techniky je možnost zobrazit objem dat přenesených mezi existujícími linky a tedy naplnění hlavního účely navrhovaného nástroje. Nejedná se o vyobrazení nové vizualizace pro každou dvojici adres, nýbrž zobrazení všech existujících dvojic/spojů najednou.
  Váha: 10

- *Možnost současného zobrazení velkého množství na sobě nezávisejících sítí*
  Vzhledem k charakteristice dat je vhodné paralelně zobrazovat informace o velkém množství sítí obsahujících menší počet adres. Toto lze provést buďto vykreslením množství nezávislých vizualizací vedle/pod sebe nebo zobrazením více sítí uvnitř jedné vizualizace (tato možnost poskytuje lepší celkový přehled a tedy za ni budou udělovány vyšší body).
  Váha: 9

- *Přehlednost pro zvyšující se množství toků*
  Každá vizualizační technika má své limity, kdy již zobrazovaná data nebudou přehledná, některé však dokáží ukázat mnohem více, než se k tomuto bodu dostanou. Pro některé případy, kdy se pohybujeme v menším množství dat také mohou některé techniky být přehlednější než ty, které lépe zvládají množství vysoká. Tato vlastnost je díky předpokládané charakteristice námi zobrazovaných dat (velké množství menších sítí) žádaná.
  Váha: 7

- *Přehlednost pro zvyšující se množství adres*
  Podobně jako v předchozím bodě, měříme schopnost přehledně zobrazit informace se zvyšujícím se množství IP adres, mezi kterými data potečou. Předchozí vlastnost se zabývá velkým množstvím existujících data transferů, tato vlastnost pak velkým množstvím existujících IP adres v síti (množství linků mezi nimi zde není relevantní).
  Váha: 7

- *Přehlednost detailu*
  Obtížnost zjištění IP adres k datové položce náležících, možnosti zhodnocení existence či velikosti datových spojů a celková jednoduchost spatření všech detailů bez nutnosti dodatečné akce (mouseover, filtry atd.)
  Váha: 6

- *Vhodnost k zobrazení více souvisejících sítí v jedné vizualizaci*
  V případě že se zobrazují jednotlivé sítě samostatně, není tato vlastnost natolik nutná, pokud však zobrazujeme detail obsahující více sítí, které sice mezi sebou nekomunikují, ale nějakým způsobem spolu mohou být spojené (například sítě 192.168.0.0/24 a 10.10.0.0/24 nacházející se na jednom stroji, kde může být žádané vypozorovat souvislost velikostí toků v těchto sítích).
  Váha: 5

- *Schopnost zahrnutí časové složky*
  Zhodnocení možnosti zachycení časové informace při současném udržení informací o velikosti existujících datových spojů. K problému lze přistupovat možností vytvoření samostatné vizualizace zobrazující diferenci

mezi dvěma či více časovými body nebo přímo zakomponováním časového faktoru do vizualizace samotné (tato možnost je hodnotnější).
Váha: 4

- *Provázanost zobrazení interních a externích (internet) toků / schopnost zobrazit tyto informace v nějaké formě najednou*
  Poskytnutí informací o velikosti zatížení veřejných IP adres k síti náležících, jejich příslušnost ke specifickému stroji.
  Váha: 3

Tyto vlastnosti umístíme do tabulky, kde budou jednat jako řádky a jednotlivé techniky později budou doplněny jako sloupce. Políčka tabulky budou obsahovat hodnocení od 0 do 10 podle toho jak moc metoda splňuje daný požadavek a celkové skóre po vynásobení váhovou hodnotou rozhodne o vítězi.

Vlastnost	**Technika**
Zobrazení všech toků	
Současné zobrazení sítí	
Škálovatelnost toků	
Škálovatelnost adres	
Kvalita detailu	
Zobrazení souvisejících sítí	
Přidání časové složky	
Přidání veřejných IP	
Skóre	
Vážené skóre	

Tabulka 1.1: Vzorová tabulka pro ohodnocení vizualizačních technik

Kapitola 2

Způsoby vizualizace toku

Dříve než se začneme zabývat specificky zacílenými druhy vizualizace toků, pojďme se na problém podívat více obecněji. Jaké techniky jsou běžně používané pro vizualizaci tokových informací a co za informace to vůbec je? O tom si řekneme více v této kapitole.

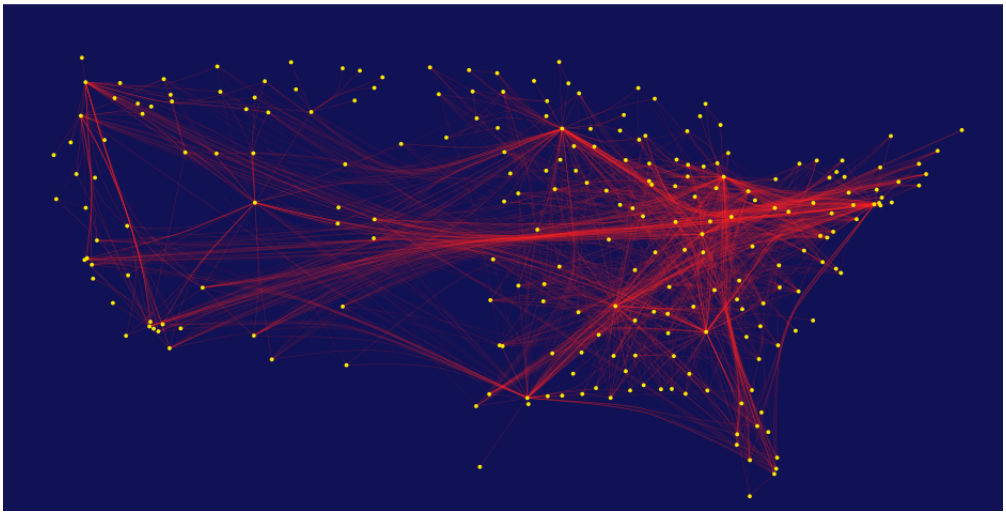
2.1 Graf / Force graf

Pravděpodobně nejzákladnějším způsobem vizualizace spojové informace je obyčejný graf. Do plátna jsou vykresleny všechny body z datové sady a pokud se mezi nimi nachází nějaký druh toku, jsou tyto body propojeny přímo či nějakým jiným druhem křivky. Indikace o velikosti toku v takovémto spoji je často vynechána, avšak lze ji doplnit manipulacemi jako jsou šířka, barva či průsvitnost spoje.
Nevýhodou této techniky je fakt, že již při malém počtu propojení vzniká značné křížení čar a rozpoznání informace může být velmi obtížné.

Tento problém se snaží řešit tzv. force graf, kde je u jednotlivých uzlů prováděna simulace vzájemných přitažlivých a odpuzujících sil a tyto body jsou pohromadě drženy pouze spoji jež mají nastavenou vyšší či menší elasticitu. Po dokončení takového fyzikální simulace dostáváme graf jehož uzly jsou co možná nejvíce rozprostřeny za účelem co nejmenšího možného křížení.

2.2 Edge bundling

Rozšíření výše uvedené techniky grafů, tato metoda je (force) graf ve kterém je namísto rovných linek pro vytvoření spojů použito křivek, které tvoří více či méně velké shluky na základě jejich počátku a konce (obrázek 2.1).



Obrázek 2.1: Silový graf spojů amerických letových linek [3]

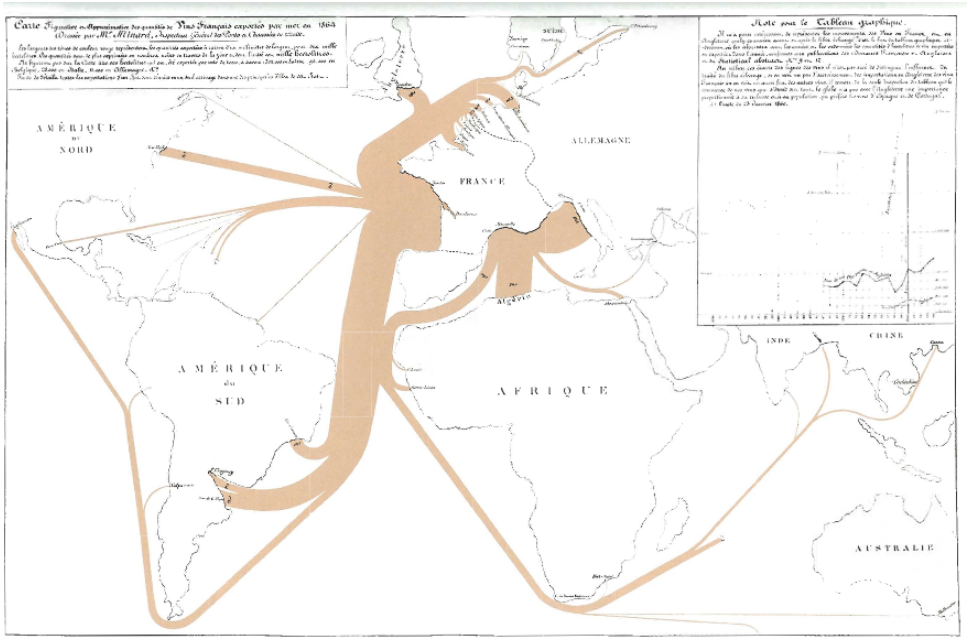
Tato vlastnost dovoluje zvýšení přehlednosti v grafu a lepší schopnost identifikace uzlů, mezi kterými se nachází spojení.

2.3 Sankey diagram

Další z běžně vídaných technik (obr. 2.2) reprezentace informace o toku jsou tzv. Sankey diagramy. Pojmenovány po Irském kapitánu Matthew Henry Phineas Riall Sankey, tyto diagramy vykreslují spojení mezi body a nastavují šířky spojů podle velikosti toku v nich.
Převážně jsou tyto diagramy používány k vizualizaci přenosu energií či nákladů mezi jednotlivými procesy a zaměřují se na schopnost lokalizace částí grafu, které k celkovému toku nejvíce přispívají.

2.4 Chord diagram

Chord diagram (obr. 2.3) je založen na podobném principu jako předchozí dva exempláře, jeho hlavní charakteristikou však je kruhové rozložení bodů, mezi kterými dochází k nějakému druhu transferu. Přenosy a jejich velikosti jsou znázorněny "strunami", které spojují jednotlivé části. Výhoda kruhového zobrazení je větší přehlednost o existujících spojeních, toto však také platí pouze do určitého množství zdrojů a spojů, kdy nakonec začne docházet ke značnému křížení.



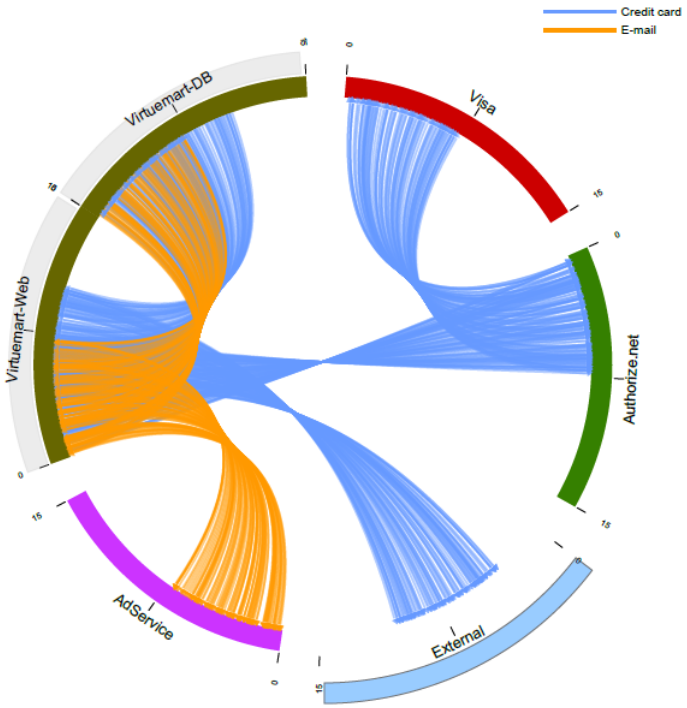
Obrázek 2.2: Mapa exportu vína z Francie, 1864 [4]

2.5 Hybridní vizualizace

Některé z těchto vizualizačních metod lze také vhodně kombinovat za účelem vytvoření hybridní techniky vhodnější pro vykreslení specifických problematik. Příkladem si uveďme například kombinaci diagramů Chord a Sankey pro vytvoření vizualizace zobrazující vliv vzdělání na eventuální kariéru, viz. obrázek 2.4.

2.6 Maticové zobrazení / binning

Vizualizací ve formě matice existuje celá řada, některé se soustředí na reprezentaci binární formy informace (políčka obsahující hodnoty ano/ne) například pro zobrazení existující spojů mezi dvěma entitami, jiné zase praktikují takzvaný binning, kde dochází k agregaci dat většího rozsahu do předem definovaného počtu buněk a jejich hodnoty jsou poté v matici nějakým způsobem vyznačeny (používané například v oblasti Big Data, obrázek 2.5).
Jednou z verzí těchto vizualizací je také tzv. heatmapa, kde jednotlivá pole matice nabývají barvy různých intenzit pro definici vztahů mezi položkou daného řádku a sloupce.

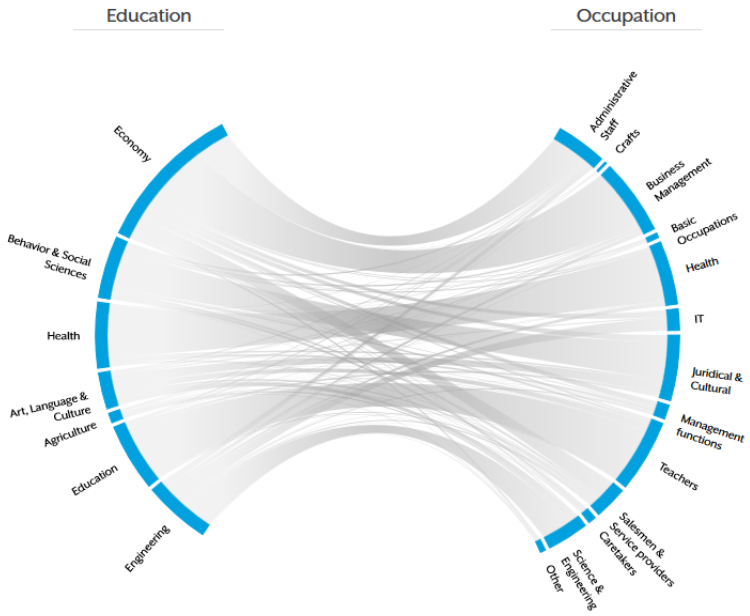


Obrázek 2.3: Přehled transakcí uživatelů [5]

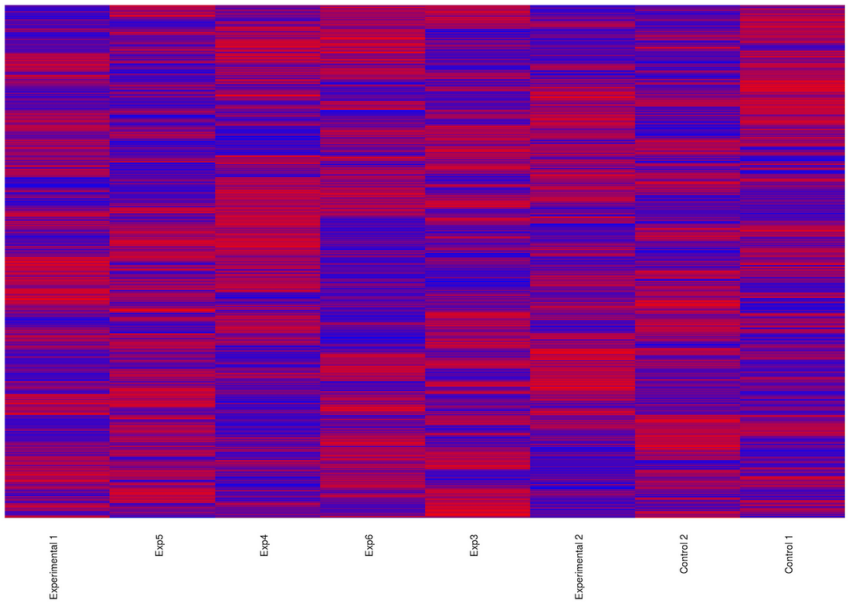
2.7 Hive

Metoda využívající mapování různých vlastností na variabilní počet os (nejméně 3) rozmístěných kruhově kolem středu. Pro každou položku dat jsou poté vytvořena spojení mezi příslušnou hodnotou dat osy a položky. Touto metodou lze vypozorovat určité typy chování a mezní hodnoty a je vhodná zejména pro širší přehled o vlastnostech použitých dat. Nevýhoda spočívá v nedostatku zobrazeného detailu jednotlivých uzlů/adres, tudíž nelze jasně vypozorovat která část vizualizace náleží které části dat.

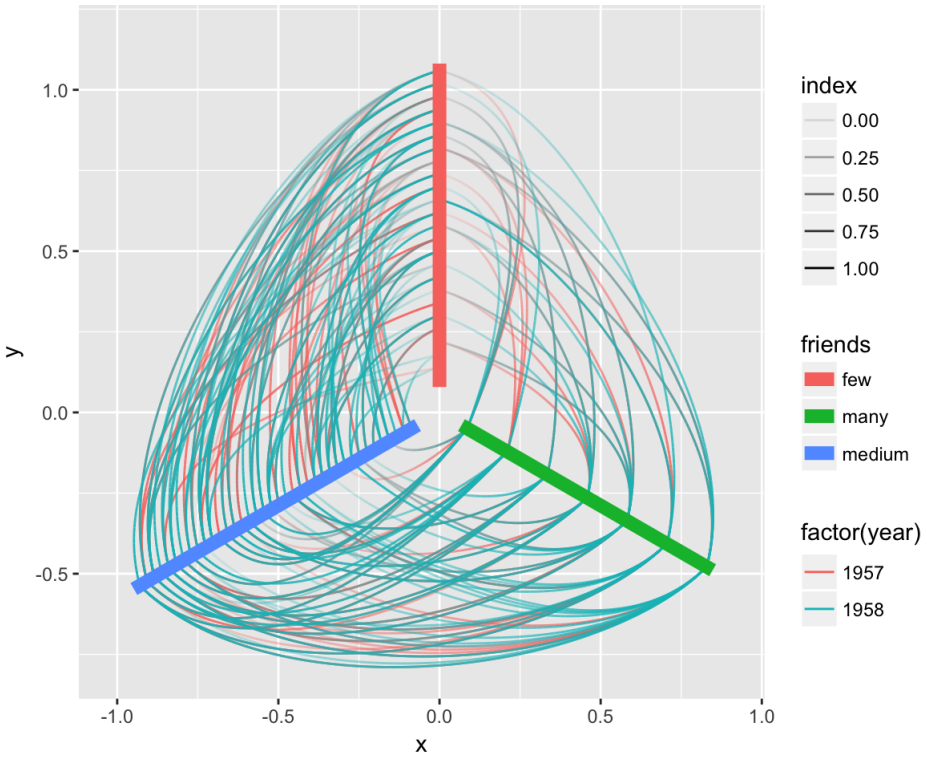
V další části práce se podíváme na to, jakým způsobem lze tyto a jiné techniky použít pro vizualizaci informace toku v počítačových sítích a jaké způsoby vizualizace používají již existující moderní aplikace pro tento účel stvořené.



Obrázek 2.4: Diagram mezinárodní migrace [6]



Obrázek 2.5: Příklad heatmapy pro Big Data [7]



Obrázek 2.6: Příklad vizualizační techniky hive nástrojem ggraph [8]

Kapitola 3

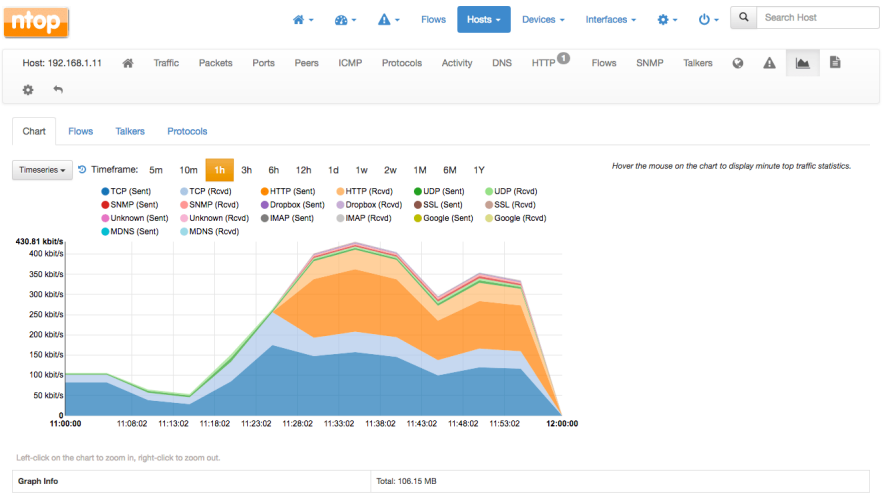
Analýza současných vizualizačních aplikací

Přes relativně raný věk oboru existuje na trhu velké množství více či méně kvalitních nástrojů pro monitoring, analýzu a vizualizaci síťových prostředků, ať již v síti cloudové aplikace nebo více klasických případech datových přenosů. Velké množství těchto nástrojů využívá ke své funkčnosti nějakou verzi v úvodu zmíněného protokolu NetFlow od firmy Cisco, nebo nějaké jeho odnoše, podíváme se však i na aplikace které k problému přistupují odlišným způsobem. Závěrem této kapitoly bychom měli mít dostatečný přehled o stavu současného vývoje, populárních vizualizačních technikách a jejich vhodnosti k použití v našem případě.

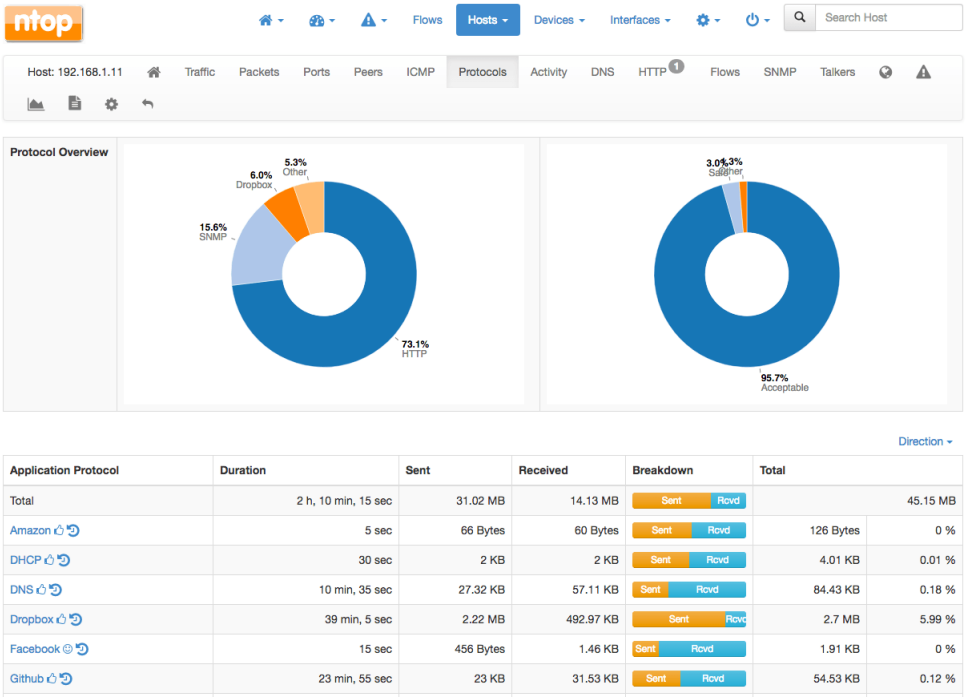
3.1 ntop

Jako první z příkladů si ukážeme projekt ntop, podporovaný pro systémy Unix, Windows a MacOSX. Jedná se o kolekci nástrojů využívající protokolu NetFlow ke sběru, analýze a zobrazení síťových informací. Tento nástroj poskytuje množství možností vizualizací, třídění a další práce s nasbíranými daty za pomocí webového rozhraní.
Soustředí se především na detailní zobrazení celkového provozu hostů v síti, součástí této komunikace, protokolů a mnoho dalších, tyto informace jsou však agregované a z většiny poskytované na bázi přehledu pro jednoho hosta.

Ačkoli je rozhraní bohaté na detaily samostatné komunikace mezi dvěma stroji (viz. příklady na obrázcích 3.1 a 3.2), jeho schopnosti zobrazení celkového stavu sítě a toku dat poskytují velmi málo až žádné vizuální informace (obr. 3.3). V příkladech použité techniky vidíme využití klasických donut, stacked area a bar diagramů.



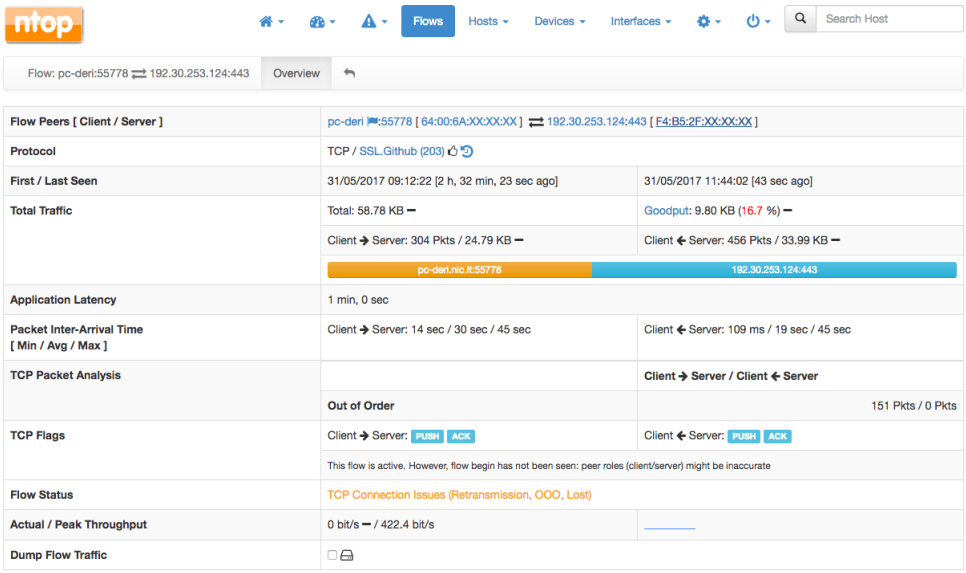
Obrázek 3.1: ntop - Historie aplikačních protokolů pro jednoho hosta [9]



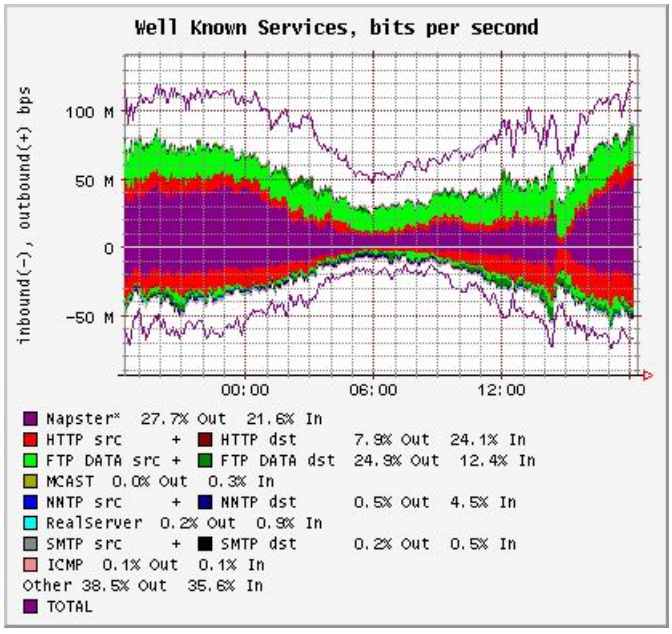
Obrázek 3.2: ntop - Detaily použitých protokolů [9]

3.2 FlowScan

S počátky v roku 2000, flowScan je jeden ze starších nástrojů k analýze a vizualizaci informací o toku dat exportovaných routery. Jedná se o jednoduché



Obrázek 3.3: ntop - Detaily toku dat [9]

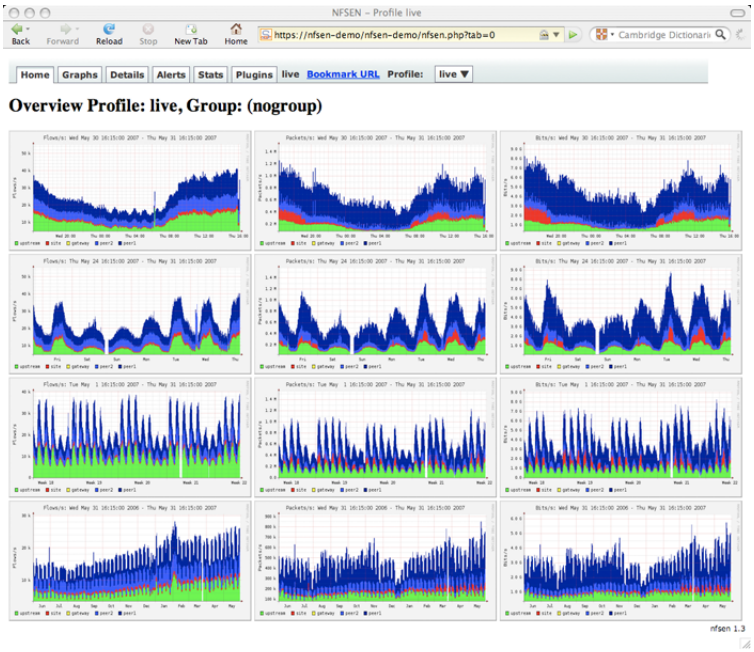


Obrázek 3.4: FlowScan [10]

obrázky vytvořené pomocí jazyka Perl a jeho modulů, které zobrazují velikosti datových toků pro jednotlivé protokoly a to jak pro upload tak pro download. Jednoduchost vizualizace je vhodná pro přehled toku tak v síti jako jejich sumu přes všechny hosty, případně pro zobrazení informací o jediném stroji,

neposkytuje však žádné informace o tom, jaký host se na výsledku v jaké míře podílí ani o toku dat mezi stroji samotnými.

3.3 NfSen

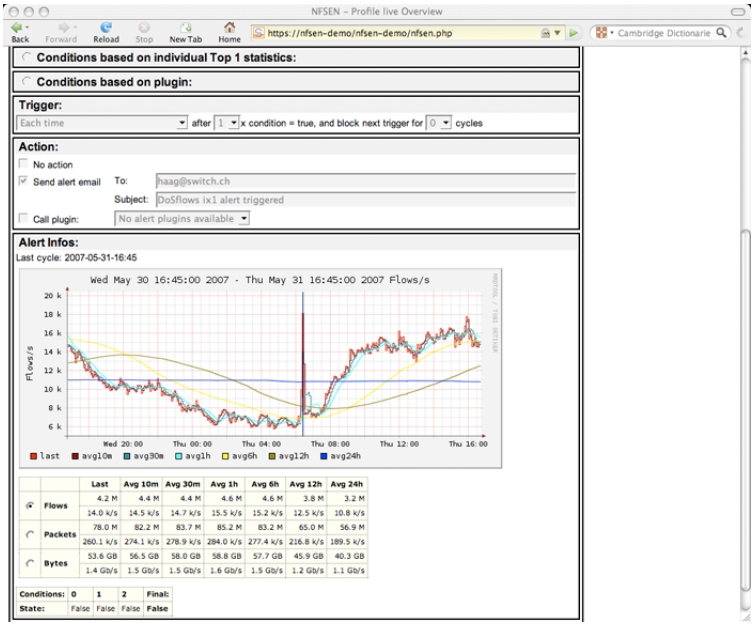


Obrázek 3.5: NfSen - Overview [11]

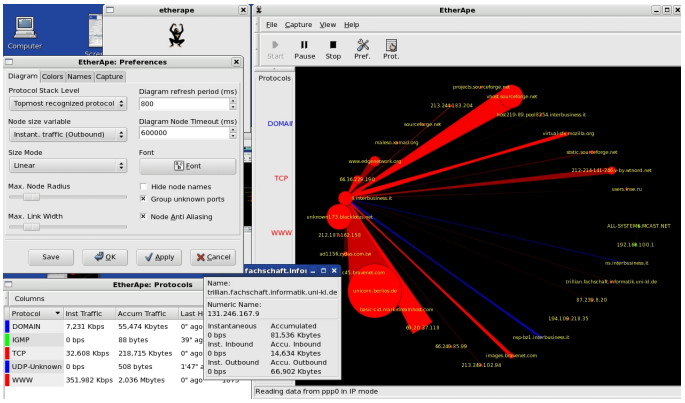
Podobně jako v případě FlowScanu, NfSen zobrazuje nasbírané (a případně dále zpracované) informace za pomocí jednoduchých grafů pro jednotlivé případy.
Jedná se o frontend webové rozhraní k přidruženému sběrači dat nfdump [12], schopného odchytávání, agregace a dalšího zpracování informací zasílaných pomocí protokolů NetFlow, IPFIX či sFlow. Na obrázcích (3.5, 3.6) opět vidíme schopnost zobrazit informace pro samostatné stroje/spojení případně sítě jako celku, nástroj však postrádá jakékoli souhrnné informace o množství toku dat mezi jednotlivými hosty.

3.4 EtherApe

Samostatně spustitelný nástroj se schopností monitoringu zadaného síťového rozhraní a vizualizace zachycených informací. Tato aplikace se od předchozích příkladů liší několika podstatnými věcmi.
První zajímavostí je to, že nevyužívá žádné další po síti rozprostřené nástroje



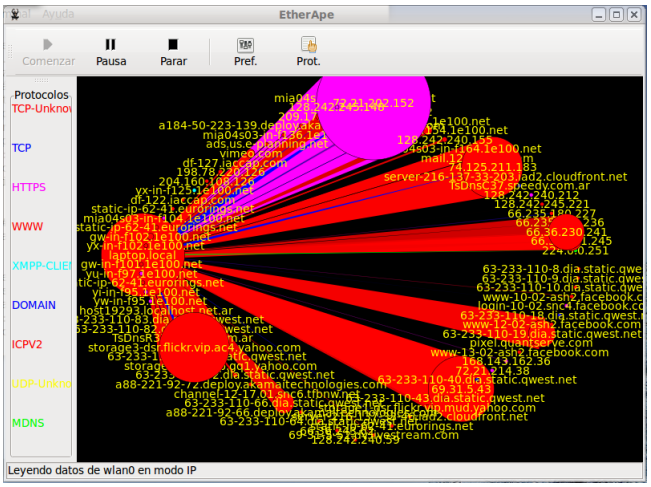
Obrázek 3.6: NfSen - Flows [11]



Obrázek 3.7: EtherApe [13]

ke sběru informací, ale funguje lokálně na sledovaném stroji. To znamená, že pomocí ní nelze sledovat informace o vícero hostech najednou, ale pouze o tom, kde nám tento nástroj běží.
Podstatně zajímavější však je způsob zobrazení informací, který EtherApe používá. Všechny zaznamenané IP adresy (či jejich přeložené webové adresy), včetně sledovaného počítače, jsou rozprostřeny do kruhu a spojení mezi sledovaným počítačem a připojenými stroji jsou vykreslena barevnými pruhy. Tyto pruhy jsou zakončena kruhem, jehož velikost je přímo úměrná velikosti přenesených dat v závislosti na celkové velikosti zachycené komunikace (obrázek

3.7).
Tento typ vizualizace je svou přehledností velice zajímavý, zejména pro malé množství informací, jeho užitečnost však začíná být pochybnější v případech velkého vzorku zobrazovaných dat (viz. obr. 3.8)

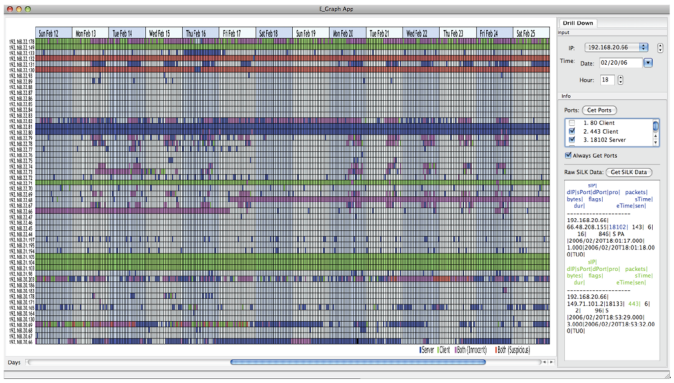


Obrázek 3.8: EtherApe - Velké množství dat [13]

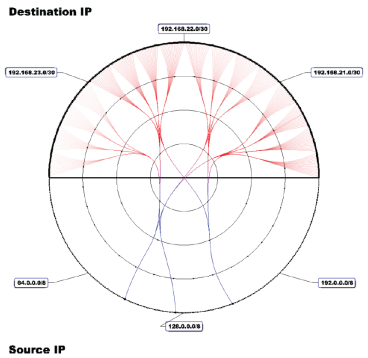
3.5 FloVis

FloVis poskytuje sadu vizualizačních nástrojů k analýze dat nasbíraných pomocí NetFlow nebo podobného protokolu. Schopnosti této aplikace jsou na první pohled zajímavější než obyčejné grafy použité v některých jiných nástrojích, z vybraných si zde ukážeme dvě.
První je metodika vizualizace toku dat pomocí **"matice"binů** využívající agregaci dat (obrázek 3.9), která vyobrazuje velikost toku dat na všech IP adresách sítě v závislosti na čase. Tímto lze vypozorovat chování dat na dané adrese v nějakém časovém okamžiku a teoreticky by mohlo být možné i vidět vývoj chování v čase, nelze však spatřit jakékoli závislosti adres na sobě a velké množství dat může dělat výsledek nepřehledným.

Druhým příkladem z této aplikace je zobrazení toku dat ze zdrojových do cílových sítí za pomocí **kruhového diagramu** (obr. 3.10). Tyto rozdělují kruh do dvou polovin, kde jedna je použita jako zdroj, druhá jako cíl. Jedná se však o hrubý přehled, obrázek však postrádá jak detailní informace o specifických adresách, tak jakékoli znázornění velikostní složky.



Obrázek 3.9: FloVis - Binning matice spojení [14]

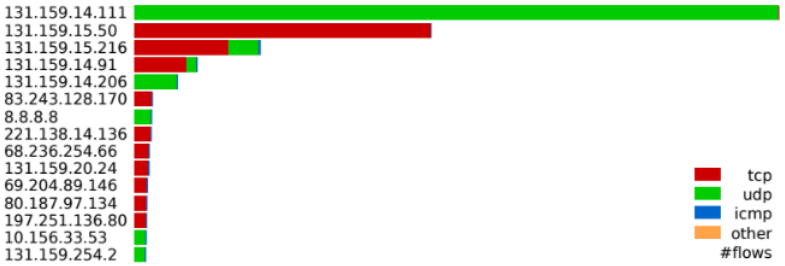


Obrázek 3.10: FloVis - Detailnější toky [14]

3.6 Flow-Inspector

Pravděpodobně pro naše účely nejbližší z existujících řešení je právě Flow-Inspector. Jedná se o webovou aplikaci vytvořenou za účely vizualizace toku dat z roku 2012. Vizualizační frontend nástroje je postaven na modulární bázi, lze do něj proto jednoduše implementovat vlastní vizualizační techniky podle potřeb uživatele, zde se však blíže podíváme na tři z již existujících modulů.

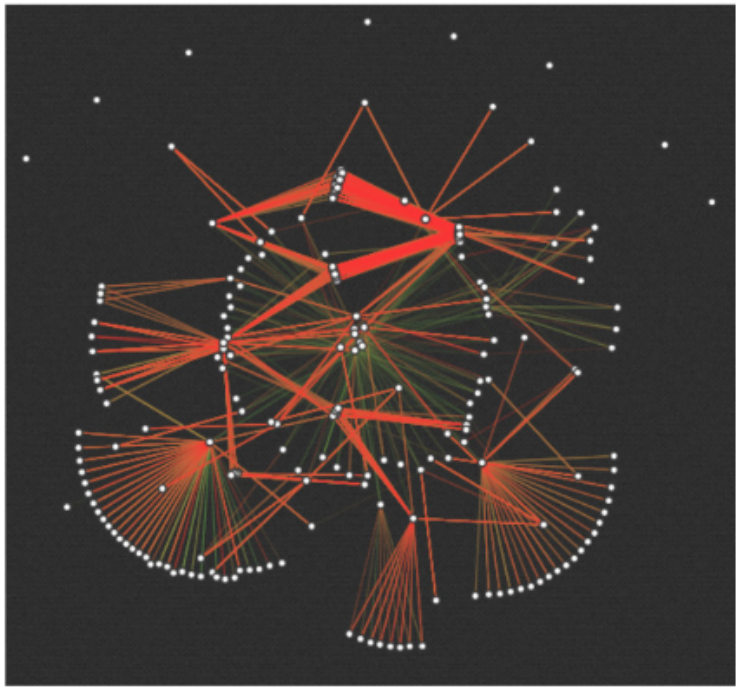
Top (obr. 3.11) - Ne nepodobný jednomu z elementů ukázaném v obrázku ntop, tento modul se chová podobně klasickému linuxovému nástroji "top". Zobrazuje detaily komunikace jednoho specifického hosta s určitým množstvím IP adres, kde se vyskytují nejvyšší datové přenosy (v příkladu 15 nejvíce datově náročných spojení). Rozšíření oproti klasickému bar diagramu se nachází v rozdílných barvách sloupce pro rozdílné protokoly, na první pohled je tedy vidět, který internetový protokol daného spojení konzumuje nejvíce dat.



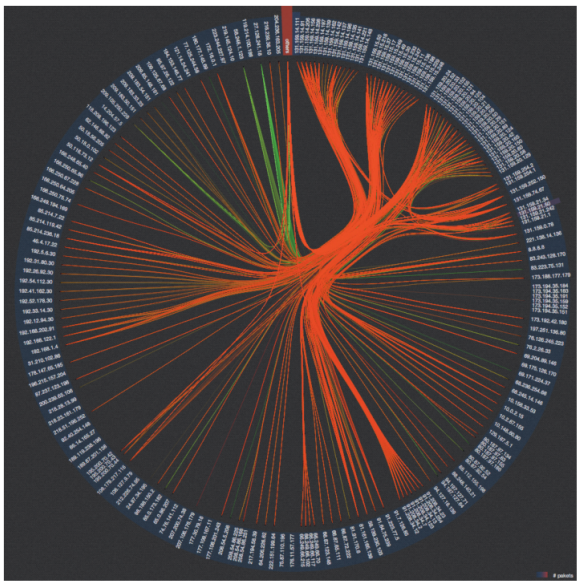
Obrázek 3.11: Flow-Inspector - Top 15 [15]

Modul využívající force graph (obr. 3.12) - Tato technika byla detailněji popsána v kapitole 2. Zde ukázaná implementace navíc podporuje zobrazení časové složky ve formě barev, nejnovější spojení jsou tedy zobrazena tou nejtmavší červenou zatímco ty, které jsou blízko úplnému zmizení ze stanoveného časového okna (takovou dobu nebyly využity) jsou zobrazeny světle zeleně.
Tento vizualizační modul je bohužel efektivní jen do určitých velikostí sítí jelikož s rostoucím množstvím uzlů budou výsledky nevyhnutelně méně a méně přehledné.

Modul využívající edge bundling (obr. 3.13) - zde je využíván podobná přístup založený na force grafu jako byl popsán v kapitole 2, nicméně je tato varianta upravena do kruhové podoby.
Jedná se o moderní a populární přístup k vizualizaci sítí a podobných problémů, jednotlivé adresy jsou rozprostřeny po obvodu kružnice, rozděleny do relevantních bloků pro přehlednost a hrany jsou vykresleny mezi existujícími spojeními. Díky vlastnosti shlukování hran které vedou do stejných či podobných cílů se uprostřed kružnice vytváří jistá přehlednost, která pomáhá s navigací.
Tato metoda nabízí vyšší přehlednost jednotlivých spojení pro větší množství dat, jelikož roztáhnutím adres po obvodu kružnice získáváme větší využitelný prostor než-li při využití pouze vertikální/horizontální složky a celkový prostor pro hrany uvnitř kružnice je také větší než při využití klasického grafu.
Stinnou stránkou techniky může být slabší informativnost o hierarchii spojení, záleží však na velikosti relevantnosti této informace a velikosti dat, při které by tato šla stále zobrazit přehledně metodikou jinou.

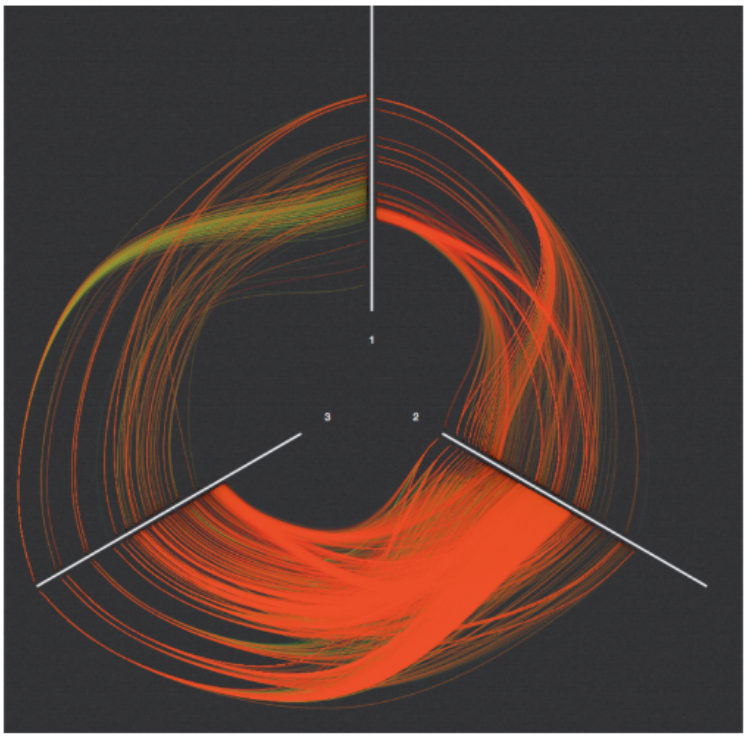


Obrázek 3.12: Flow-Inspector - Force Graph [15]



Obrázek 3.13: Flow-Inspector - Edge Bundling [15]

Hive (obr. 3.14) - Posledním z příkladů schopností této aplikace je technika Hive. V obrázku vidíme, že se jedná o dobrý způsob prezentace pro vyčtení celkových trendů chování dat, metoda však postrádá schopnosti zobrazení detailnějších informací.



Obrázek 3.14: Flow-Inspector - Hive [15]

3.7 Shrnutí poznatků

Různých implementací a přístupů k problému ve světě existuje nespočet, v této kapitole jsme si ukázali exempláře alespoň některých z nich a prozkoumali metodiky, které ke své funkci využívají. Mezi ty patřily následující

- Bar chart
- (Stacked) area chart
- Donut chart
- 2D graf / multigraf
- Matice binů či dat

- Kruhové zobrazení
- Force graf
- Hive

a zajisté bychom mohli nalézt ještě další (jako např. 3D graf). Lze tedy vidět, že pro zobrazení informací o toku dat lze využít široké spektrum metodik, podstatné je, co se snažíme zobrazit.

Kapitola 4

Ohodnoceních vybraných vizualizačních technik

V této chvíli již můžeme na základě předchozí analýzy přejít k bodovému ohodnocení vybraných kandidátů, tak, jak jsme si ho definovali v kapitole 1.

V rychlosti si nejprve připomeňme zkoumané vlastnosti a jejich váhy (v závorkách) pro celkové hodnocení.

- Zobrazení všech toků (10)
  Úroveň schopnosti vizualizace vykreslit informace o všech existujících tocích v datech.
- Současné zobrazení sítí (9)
  Možnost vizualizace zobrazit větší množství menších sítí (průměrně malé množství IP adres v nich).
- Škálovatelnost toků (7)
  Jak moc zůstává vizualizace přehledná s rostoucím množstvím tokových dat.
- Škálovatelnost adres (7)
  Zachování přehlednosti vizualizace při zvyšujícím se množství IP adres v síti.
- Kvalita detailu (6)
  Úroveň jednoduchosti spatření detailních informací bez dodatečných akcí (filtry, najetí myší...)
- Zobrazení souvisejících sítí (5)
  Schopnost zobrazení korelace toků mezi několika sítěmi.
- Přidání časové složky (4)
  Zhodnocení možnosti zachycení časové informace.

- Přidání veřejných IP adres (3)
  Možnost integrace informací o zátěži na veřejných IP adresách.

Tyto vlastnosti a z předchozích kapitol vybrané kandidáty vizualizačních technik si umístíme do tabulky a každou z nich bodově ohodnotíme na stupnici od 0 do 10.
Bodové ohodnocení každého z políček bude nakonec vynásobeno příslušnou vahou vlastnosti a tyto hodnoty sečteny. Techniky dosahující nejvyššího skóre poté budou ty, jež jsou nejvhodnější k námi požadovaným účelům.

Vlastnost	**Bar Chart**	**Area Chart**	**Donut**	**Matice**	**Kruh**	**Force Graph**	**Hive**
Zobrazení všech toků	0	0	0	10	9	9	8
Současné zobrazení sítí	0	0	0	7	9	9	9
Škálovatelnost toků	7	1	5	10	7	4	7
Škálovatelnost adres	7	1	3	7	9	6	9
Kvalita detailu	8	5	6	8	8	6	4
Zobrazení souvisejících sítí	0	0	0	7	8	8	6
Přidání časové složky	1	8	0	4	4	4	6
Přidání veřejných IP	0	0	0	5	5	5	5
Skóre	23	15	14	58	59	51	54
Vážené skóre	**150**	**76**	**92**	**396**	**402**	**341**	**366**

Tabulka 4.1: Ohodnocení vizualizačních technik

Jak vidíme v tabulce 4.1, některé z analyzovaných metodik jsou pro naše cíle podstatně vhodnější než jiné a některé naopak zcela nevhodné (bar chart, area chart, donut chart). Jako velmi kvalitní se ukázali čtyři ze sedmi testovaných metod, jimiž jsou force graf, hive, matice a kruhový graf, poslední dvě jmenované z nichž jsou jasně na vrcholu hodnotící tabulky kde spolu velmi blízko soupeří.
Mezi hlavní nevýhody silového grafu patří špatná škálovatelnost spolu s rostoucím množstvím dat, kde se graf nakonec stává velice nepřehledným a nelze zamezit křížení. Hive naopak nemá problém s velkým množstvím hodnot, ale postrádá ve schopnostech zobrazení detailních informací, výsledek této metody je zejména zajímavý jako celek.
V dalších úvahách a samotném návrhu se tedy budeme zabývat zejména technikami založených na *kruhovém* a *maticovém* zobrazení.

Kapitola 5

Architektura vizualizační aplikace

V této kapitole se podíváme na to, z jakých funkčních celků se vizualizační aplikace skládá a tedy na které části se při implementaci bude třeba zaměřit, jejich specifikace a případné další poznatky, které bude třeba brát v úvahu při samotném vývoji.
Také se pokusíme určit, kolik různých vizualizací bude v nástroji pravděpodobně nutno použít k dosažení požadované funkcionality.

5.1 Volba platformy

Přestože se zde ještě nebudeme zabývat specifickou technologií, k přesnějším úvahám o architektuře je již nutné zvolit si platformu, na které se bude aplikace nacházet. Budeme-li uvažovat pouze klasické počítačové prostředí (jelikož mobilní verze aplikace nás v tuto chvíli nezajímají) nabízí se nám tři hlavní možnosti

- *Microsoft Windows*
  Nejrozšířenější operační systém na světe, zaměřen na běžné uživatele a nacházející se téměř na každém pracovišti, tato platforma nabízí možnosti široké použitelnosti aplikace spolu s množstvím podpůrných knihoven a dalších nástrojů vytvořených buďto komunitou nebo nějakou z jiných firem.
  Aplikace založené na tomto systému, jakožto klasické desktopová aplikace, získávají zejména velké robustnosti a přístupu k většině systémových prostředků. Jedná se o prostředí vhodné k běhu programů potřebujících velké výpočetní a paměťové zdroje.
  Nevýhodou je vyšší komplikovanost vývoje způsobená uzavřeností systému, ne vždy vhodně navržených či užitečných protokolů a nekompati-

bilita s mnoha užitečnými nástroji (které jsou častěji zaměřeny na OS Linux).

- *Linux OS*
  Druhé z hlavních desktopových prostředí nabízí mnoho různých variant operačních systémů, z nichž převážná většina staví na své otevřenosti a přívětivosti vůči zkušenému uživateli a/nebo programátorovi. Pro platformu je vyvíjena celá řada nástrojů, většina z nichž se řídí otevřenou Linuxovou politikou a je zdarma k použití. Jednotlivé distribuce a protokoly systému jsou mezi sebou většinou vysoce kompatibilní, což však nelze vždy říci o kompatibilitě s OS Windows.
  Nevýhoda systému je zcela opačná oproti Microsoft Windows, tedy vyšší složitost běžného používání pro nezkušeného uživatele.

- *Webová aplikace*
  Poslední ze tří nabízených možností se vyznačuje zejména maximální rozšířeností a nezávislostí na operačním systému na stroji se nacházejícím. Aplikace takto vytvořené pro svůj chod vyžadují pouze jakýkoli moderní prohlížeč (s možnou výjimkou prohlížeče Internet Explorer) a jejich funkčnost je poté zaručena na téměř jakémkoli systému, včetně mobilních (ačkoli tam může být vyžadována dodatečná optimalizace pro akomodaci rozlišení). Využití moderních programovacích a skriptovacích jazyků také aplikaci dovoluje vysokou flexibilitu, snadnou tvorbu animací a téměř neomezenou kreativní svobodu (css, javascript atd.)
  Stinná stránkou platformy jsou její omezené možnosti práce se zdroji systému, které ve výsledku nabízí daleko menší výpočetní výkon než desktopové řešení. Také je třeba počítat s množstvím odlišných prohlížečů, které nemusejí vždy nabízet stejné možnosti a kompatibilitu s použitými technologiemi (například neobvyklý nebo zastaralý prohlížeč).

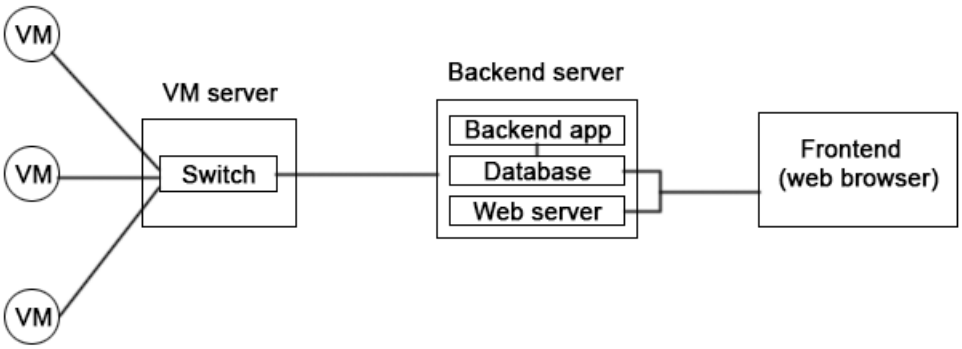
Pokud to kompatibilita použitých technologií dovoluje, nemusí se jednotlivé části aplikace nacházet na stejných platformách, což je bezesporu výhoda, která nabízí využití silných částí jednotlivých systémů pro zamýšlené úkony. Po zvážení možností výše spolu se současnými zkušenostmi autora s jednotlivými nástroji a úmyslem vysoké kompatibility byly nakonec zvoleny tyto platformy:

Backend - OS Linux
Současné servery zde uvažovaného cloudu využívají pro svůj chod operačního systému Linux, stejně tak jako většina ostatních ve světě existujících řešení, bude tedy z tohoto systému nutné nějakým způsobem získávat data o tocích v sítích. K tomuto účelu bude teoreticky možné použít externích nástrojů ke sběru dat, které jsou pro systém Linux vyvinuty.K zajištění možnosti jejich využití a ulehčení vývoje (práce s databází, programování, spouštění a monitorování backendové aplikace atd.) se tedy tato možnost jeví jako nejlepším

řešením.

Frontend - Webová aplikace
Takto vytvořený nástroj bude široce dostupný přes téměř jakýkoli webový prohlížeč, nebude nutné ho instalovat ani splňovat žádné další systémově specifické požadavky. Schopnosti moderních webových technologií také umožní velkou flexibilitu v oblasti tvoření animací, vizualizací a celkově esteticky dobře vypadajících výsledků.
Nižší výkonnostní schopnosti platformy by neměly být problémem, jelikož se tato část bude využívat k vykreslování již zpracovaných dat, všechna výpočetně náročná logika tedy bude přenechána backendové aplikaci.

5.2 Obecná architektura



Obrázek 5.1: Obecná architektura webové vizualizační aplikace

Jak již bylo v části 5.1 lehce zmíněno, vizualizační aplikace potřebuje mít mimo jiné zejména dvě hlavní funkční části, které na sobě mohou být částečně nezávislé. Těmi jsou:

Backendová aplikace
Hlavní výpočetní část nástroje, starající se o sbírání/příjímání/parsování a zpracovávání dat o tocích v síti dle toho jaký způsob a nástroje k získávání těchto dat používáme. Jak můžeme vidět na obrázku 5.1, předpokládá se, že

tato část aplikace běží na vlastním serveru a tedy není součástí vizualizačního modulu. Jejím úkolem je získat z dostupných informací relevantní data, postarat se o jejich případnou agregaci, transformaci do zvoleného datového formátu, přiložení časového razítka a uložení na správné místo v databázi, odkud budou tato data přístupna vizualizační části.
Dle potřeb a možností může tato aplikace poskytovat ještě další služby dle úsudku programátora.

Frontendová aplikace
Tato část aplikace je nahrávána do prohlížeče uživatele v momentě kdy ten přistoupí na patřičnou webovou stránku. Jejím hlavním úkolem je vyzvednutí specifických dat z databáze, jejich transformace a filtrování a následné namapování a vykreslení do prohlížeče. Často jsou nabízeny možnosti další interakce s tímto výsledkem ve formě filtrů, zobrazení dodatečných detailů, transformací, změny typu mapování a mnoho dalších.
Zde se také řeší všechny interakce uživatele se službami aplikace, jako jsou přihlášení, změny a ukládání/načítání uživatelských nastavení (layout, použitý css soubor, preferované filtry a hodnoty atd.).

Mimo tyto hlavní dvě součásti se ve struktuře ještě nacházejí další podpůrné části, těmi obecně jsou

Databáze
Skladiště dat a informací, využíváno jak front tak backendem. Zde jsou ukládány zpracovaná a připravená data, informace o uživatelích, jejich preference a jakékoli další informace, které je nutno nějakým způsobem uchovat. Vzhledem k webové charakteristice vizualizační části je toto jediný způsob jak je možno permanentně uchovávat informace.
V prezentovaném obrázku obě části aplikace využívají stejné databáze z důvodu zachování jednoduchosti, pokud by tomu však bylo potřeba může každá součást používat databázi vlastní, třeba i úplně odlišného typu a architektury.

Webový server
Samostatná aplikace zastávající úlohu obsluhy webových připojení k serveru a nahrání kódu frontendu do prohlížeče. V závislosti na typu použité technologie může toto být všechno co se od serveru požaduje, nebo naopak může vykonávat ještě další výpočetní úkoly namísto webového prohlížeče.
V navrhované architektuře se server nachází na stejném fyzickém stroji jako backendová část aplikace, což zajišťuje jednodušší a bezpečnější práci s databází, která je také na stejném stroji, není tomu však potřeba a může se nacházet na libovolném místě (pokud bude schopný připojit se k databázi).

Sběrač dat
V tuto chvíli ještě blíže nespecifikován, jelikož se jedná o technologicky specifickou část, zajišťuje však záznam informací o tocích na switchi, jejich částečné zpracování a filtrování a přepravení těchto dat do backendu k dalšímu zpracování a uložení.
K účelu slouží různá řešení od skenování paketů procházejících rozhraním serveru, využití služeb a protokolů switche/routeru nebo například vlastní skript či program přímo interagující s backendem.

5.3 Rozlišení sběru dat

Sledovaná data můžeme získávat v několika variacích, které různě zvyšují/zmenšují výpočetní a paměťovou náročnost jejich zpracování a uchování, ale také možnosti detailního filtrování, které mohou být implementovány. Obecně platí, že čím více prostředků je potřeba ke zpracování a uchování dat, tím vyšší rozlišení nám bude poskytnuto. Možnosti jsou:

Uchování všech dat
Naivní a přímočaré řešení, které poskytuje absolutní svobodu se zpracováním. V této formě poskytujeme uživateli kompletní informace o tom, co se v síti dělo v jaký časový okamžik s kterým paketem. Problém zde však nastává s obrovským množstvím dat, které by bylo nutné uchovávat. Každou hodinou serverem můžou protéci miliony či miliardy paketů, pokud řekneme že informace o každém bude jeden řádek souboru, již za jeden den se nacházíme v oblasti Big Data, která vyžaduje mnohem vyšší fyzické prostředky než nám budou v tomto projektu dostupné. Navíc lze říci, že se jedná o neúměrně vysokou nevyváženost v porovnání investice/benefit a tedy tato možnost zde není vhodná.

Agregace
Druhou možností je agregace dat časových úseků. Všechny pakety jsou stále sledovány a případně zaznamenány, dochází však k součtu počtu paketů stejného typu (zdrojová a cílová IP adresa, protokol) zaznamenaných v určitém časovém okně a pouze tato informace je nakonec uložena. Namísto statisíců řádků dat pro okno jedné minuty tedy nakonec získáme například pouze několik desítek. Podstatnou volbou zde je správné nastavení velikosti časového okna. Velké intervaly znamenají vysoké úspory a nízké výpočetní náklady ke zpracování vizualizační částí, možnosti zobrazení detailů a přesnosti zobrazovaných informací jsou však omezené. Volba závisí na aktuální potřebě aplikace či uživatele, v našem případě se bude pravděpodobně jednat o řádově jednotky minut.

Sampling
Způsobem ukládání dat velmi podobné řešení jako agregace, rozdíl však nastává v tom, že zde nejsou sledována a zpracovávána všechna data, která zařízením procházejí. Využívá se tzv. sampling rate, tedy časového intervalu na konci kterého se provede jeden záznam stavu paketů v zařízení. Takto nasbírané vzorky se průměrují a získává se tím odhad chování toků v síti. Metoda je výpočetně velmi přívětivá, jelikož provádí nějakou činnost pouze jednou za každých několik sekund, přesnost měření je však velmi ovlivněna, jelikož získaná data nereprezentují to co se v sítí skutečně dělo, nýbrž to co se sběrač domnívá se v ní dělo. Mnohokrát se může jednat o přijatelně přesné odhady, rozlišení této metody je však zdaleka nejmenší.

Z důvodu zachování kvality rozlišení dat bez využití extrémních prostředků k jejich uchovávání byla nakonec zvolena možnost střední cesty, tedy *agregace*.

5.4 Nezbytné informace k zobrazení

Z rozboru požadavků na funkčnost aplikace lze vyvodit informace, které budeme od našeho nástroje potřebovat nějakým způsobem zobrazit. Pokud to bude možné, lze tyto celky alespoň částečně zobrazit v jedné vizualizaci, lze je však rozdělit na jednotlivé části zobrazitelné samostatně.

- *Celkový přehled sítí s náčrtem jejich vytížení*
  Jedná se o "overview"sítí cloudu a to iděálně všech, pokud to bude jejich množství umožňovat.
  Mimo počtu sítí a množství adres, které obsahují by mělo být možné alespoň přibližně vidět existující a neexistující linky mezi adresami a množství dat, které mezi nimi protéká. K nápomoci s detekcí pouze "zajímavých"částí (například kriticky velký tok) mohou sloužit filtry k odstranění nezajímavých částí nebo například vhodně zvolená barevná škála navržená ke zvýraznění mezních hodnot.

- *Detailní pohled na toky jedné sítě*
  Ať již po využití předchozí části k detekci potencionálně zajímavé sítě nebo použití ručního vyhledání specifické sítě je vhodné uživateli zobrazit plný detail spojů mezi adresami, velikosti toků v co nejlepším možném rozlišení, výpisem adres v síti se nacházejících a další dodatečnou funkcionalitou (například vyfiltrování pouze relevantních toků při přejezdu myší na nějakou specifickou adresu, další detailnější filtry atd.). Tato část také může obsahovat nějaký alespoň přibližný přehled o aktivitě veřejných IP adres.

- *Pohled na aktivitu veřejné IP adresy sítě*
  Vizualizačně zcela odlišnou techniku bude téměř jistě vyžadovat detailní

zobrazení aktivity na jedné či více veřejných IP adresách s přístupem do internetu. Na rozdíl od předchozích příkladů, kde se jednalo o připojení řádově nanejvýš k desítkám dalších adres, zde se jedná o vztah 1:N kde N se může pohybovat v řádu tisíců i milionů. Pravděpodobně nejvhodnější přístup k problému je tedy celková agregace všech dat na těchto adresách protékajících, čímž sice ztratíme informace o tom k jaké z vnějších adres jednotlivé části celku náleží, nabízená komprese datových souborů je však velice výhodná, zejména jelikož v této části není vysoké rozlišení dat požadováno.

- *Detailní pohled na toky více sítí* [volitelné]
  Dodatečná funkcionalita k implementaci pokud bude v rámci projektu čas a/nebo pokud to bude využitá technika umožňovat. Jedná se o podobný přehled jako v rámci jedné sítě, je zde však nutné sesbírat všechny virtuální sítě, se kterými daný virtuální stroj nebo ostatní stroje na jeho síti komunikují a ty poté zobrazit takovým způsobem aby bylo možné velikosti toků v nich porovnávat a vypozorovat případné souvislosti (například pokud stroj 1 zasílá velký objem dat na stroj 4 po síti 192.168.0.0 a stroj 4 posílá podobný objem dat na stroj 6 po síti 172.27.0.0, bude z tohoto pozorování zřejmé, že se pravděpodobně jedná o tentýž datový proud - přesnější informace zobrazit nelze, jelikož první typ paketu je adresován ze stroje 1 na stroj 4 a druhý typ paketů ze stroje 4 na stroj 6, co se na stroji 4 dělo, tj. jedná-li se o stejná data nebo kompletně odlišné využití sítě pro jeho vlastní účely nelze říci).

Kapitola 6

Návrh metod vizualizace a prototyp

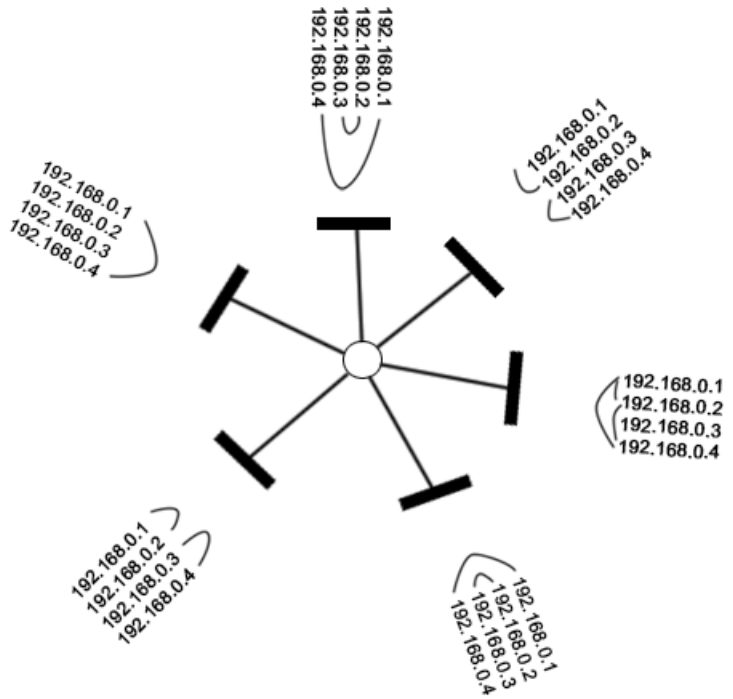
V dosavadních částech textu jsme došli k tomu závěru, že ze zkoumaných vizualizačních metod dosahují nejvyššího potenciálu ty, které jsou založené na bázi kruhového a maticového zobrazení. Nyní je tedy čas tyto techniky blíže prozkoumat a rozebrat jejich možnosti, na základě čehož bude možné vytvořit náčrt předpokládané podoby vizualizace.
S tímto náčrtem lze vytvořit prototyp využívající námi vygenerovaná dummy data k získání lepšího náhledu na chování metody v různých situacích. Získané výsledky porovnáme a probereme jejich silné a slabé stránky.

6.1 Vnitřní síť

Nejpodstatnější částí této kapitoly je návrh vizualizací zaobírající se interní částí sítě, tedy bez vyššího uvažování veřejných adres a externích IP k nim připojených. Díky blízké provázanosti zobrazení více sítí a jedné sítě samotné zde budeme probírat obě problematiky současně, což nám umožní odhadovat dopady při změnách chování přehledu na chování detailu a naopak.

Kruhové zobrazení
Viděno v příkladech použití jako způsob organizace velkého množství adres do kruhu a vykreslení spojů mezi nimi. Ta je pro lepší přehlednost možné organizovat metodou edge bundling, kategorizovat barvami nebo měnit jejich šířku a průhlednost dle jejich vlastností. Výhodnou vlastností je možnost oddělit od sebe bloky adres zavedením mezery do kruhu, čímž lze zvýraznit jejich rozdělení na jednotlivé sekce, pokud taková vlastnost v datech existuje.
Námi cílená data jsou speciální v tom ohledu, že je předpokládané velké množství takovýchto sekcí (oddělené sítě), kde každá z nich bude obsahovat spíše menší množství adres. Zarovnáním těchto k sobě náležících adres vedle sebe

a rozdělením do jednotlivých funkčních celků nám omezí nebo dokonce úplně předejde situacím křížení hran skrz střed kružnice, jelikož jednotlivé adresy se budou nacházet vedle sebe. Na rozdíl od příkladu v kapitole 3, kde existují možná spojení mezi velkým množstvím adres se nám tedy kruh rozpadá na množství nezávislých částí. Tento fakt přidává na přehlednosti spojení a uvolňuje nezanedbatelné místo uprostřed kružnice, které může být využito k jiným účelům.
Návrh řešení uvažuje využití tohoto prostoru k vykreslení agregovaných informací o vytížení veřejných IP adres ke každé síti náležících za použití částí oblouků kruhu reprezentující velikost bloku sítě (v rámci místa zabíraného vypsáním všech adres). Tento oblouk je poté rozdělen na několik sektorů, podle množství veřejných adres, každý z nichž reprezentuje jednu takovouto IP. Od každého takovéhoto sektoru povede link do středu kruhu (reprezentující bránu do internetu) jehož barva bude nastavena podle velikosti datové zátěže na této adrese (například světle zelená pro minimální zátěž, ostře červená pro extrémní zátěž).
Výsledný produkt by tedy poté vypadal (velmi zhruba) takto (obrázek 6.1).



Obrázek 6.1: Náčrt možného kruhového zobrazení přehledu sítí v cloudu

Obrázek nedisponuje diskutovaným odlišením zátěže, ani různou velikostí

spojů mezi adresami pro odlišení jejich velikostí, jedná se však pouze o náčrt k přiblížení myšlenky a využití v prototypu. Všechny sítě zde uvedené obsahují přesně čtyři privátní adresy (192.168.0.1 až 192.168.0.4) a po jedné adrese veřejné (kdyby těchto bylo více, blok by byl rozdělen na více částí, každá z nichž by obsahovala vlastní spoj do internetu).

Zobrazení detailu jediné sítě poté vypadá velmi obdobně jako příklad výše, rozdíl je pouze v tom, že je odstraněna středová část diagramu a adresy dané sítě jsou rozprostřeny po obvodu celé kružnice. V tomto případě již bude docházet k průchodu spojů středem kružnice.

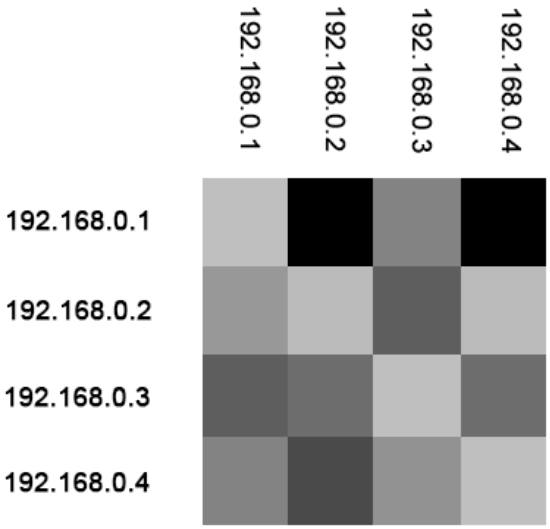
Maticové zobrazení
Příklad uvedený v kapitole 3 (obrázek 3.9) se spíše zabýval agregací dat adres do jednotlivých přihrádek a jejich vyobrazení na časové ose, než-li klasickým maticovým zobrazením N:N. To funguje na principu vytvoření jednoduché čtvercové matice, jejíž řádky i sloupce náleží ke stejným parametrům. Políčka matice jsou poté vyplněna barvou o specifickém odstínu či intenzitě, které nějakým způsobem vyjadřují vztah položky řádku k dané položce sloupce. Tento způsob vizualizace je nazýván anglickým termínem heatmapa.
Návrh vizualizace sítě tímto způsobem je tedy jednoduchý. Řádky i sloupce matice budou vyjadřovat IP adresy sítě a políčka matice poté intenzitu provozu mezi danou dvojicí (řádek/sloupec). Hodnoty políček diagonály budou nula (provoz z adresy na sebe samotnou).
Náčrt tohoto řešení vidíme na obrázku 6.2.

Tento způsob poskytuje jednoduchost a zejména škálovatelnost, jelikož lze vyhledat a přečíst informace o libovolném spoji bez rizika jeho ztráty v ostatním kontextu. Tato přehlednost zůstává s přibývajícím množstvím adres stejná a nijak se nemění.
Nevýhoda spočívá ve vyšší náročnosti na vyčtení údajů a zejména pak v nárocích na dostupné rozměry k vykreslení většího množství heatmap. Tento problém je možné částečně řešit vynecháním popisků řádků a sloupců, nastává však otázka jakou by takovéto řešení mělo informační hodnotu.

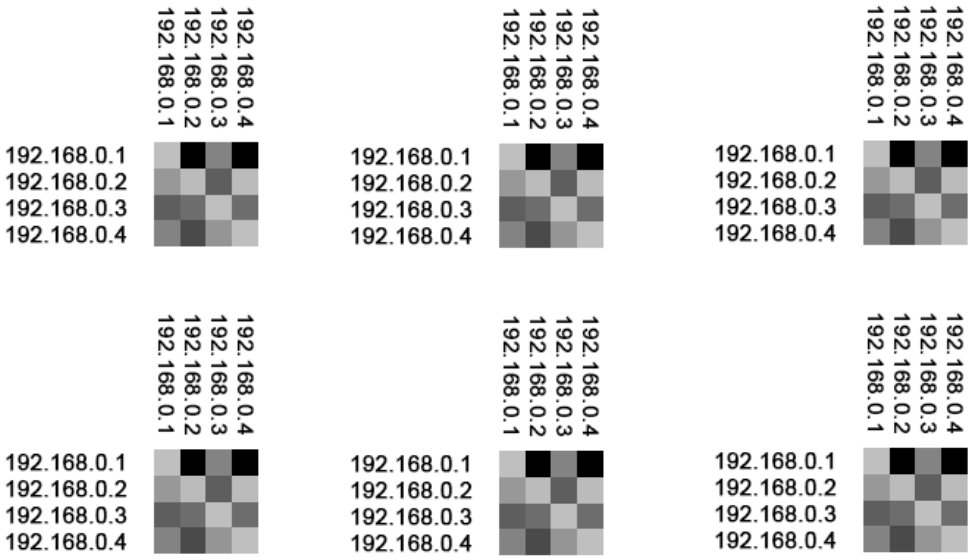
Verze pro potřeby vykreslení více sítí pro celkový přehled je téměř identická (obr. 6.3), pouze tyto mapy řadí vedle sebe. Jisté komplikace může způsobit proměnná velikost různých sítí (rozdílný počet adres), kde může nastat problém s efektivním uspořádáním map do prostoru obrazovky, tento problém však v náčrtu není uvažován (všechny sítě v obrázku se skládají ze čtyř vnitřních IP adres).

6.2 Vnější síť

V tomto úkolu můžeme efektivně využít techniku, která se ukázala naprosto nevhodnou k úkolu předešlému, kterou je bar chart.

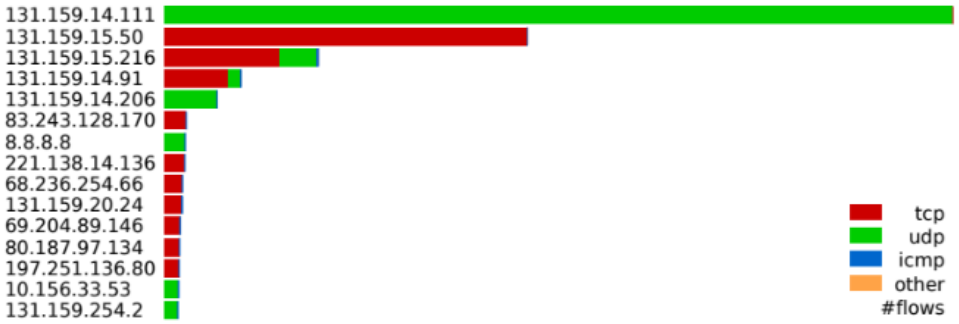


Obrázek 6.2: Náčrt detailu heatmapy



Obrázek 6.3: Návrh zobrazení více sítí pomocí heatmap

K návrhu zde není třeba vytvářet nové náčrty jelikož můžeme jednoduše využít příklad uvedený výše z aplikace Flow Inspector, který toto řešení implementuje. Informace pro každou veřejnou IP adresu sítě tedy bude možné zobrazit ve formě bar chartu a tak jednoduchým způsobem porovnávat velikosti zátěží na jednotlivých adresách a také velikost aktivity na adrese vůči nějakému limitnímu bodu (například maximální hodnota ze všech zkoumaných sítí).



Obrázek 6.4: Flow-Inspector - Top 15 [15]

Z důvodu časové úspory a předešlé existence řešení tato varianta v prototypu nebude implementována a bude se počítat s jejím použitím ve výsledné aplikaci. Pozornost bude momentálně soustředěna zejména na předchozí část vizualizace - vnitřní sítě.

6.3 Generátor dat

Před testováním samotných vizualizací je nutné mít dostupná data, ze kterých se budou tyto diagramy tvořit. Samozřejmě nejlepší variantou by bylo použití reálných dat z různých serverů v provozu (nebo alespoň v testovacím prostředí), jelikož tato data však nejsou v tuto chvíli dostupná, bude nutné si vytvořit vlastní dummy data, založená na těchto předpokladech

- Velké množství samostatných sítí
  - Vzájemná nezávislost
  - Předpokládá se, že jedna takováto síť neobsahuje další podsítě
- Menší množství privátních adres uvnitř sítě
  - Řádově jednotky, nanejvýše desítky
  - Více sítí může sdílet identické adresy
  - Možnost komunikace s jakoukoli jinou adresou uvnitř sítě
  - Nemožnost navázání spoje mimo síť

- Možnost veřejných IP adres v každé síti
  - Řádově jednotky, může být nula
  - Nemůže přímo komunikovat s adresami uvnitř sítě
  - Obsahuje pouze agregaci dat všech navázaných vnějších spojení

Tyto data by bylo možné vytvářet ručně, vzhledem k jejich potřebné rozsáhlosti a variabilitě ke kvalitnímu testování prototypů je však vhodnější vytvořit náhodný generátor, který na základě zvolených vstupních parametrů vyplní předpřipravenou šablonu. Ta je vytvořena ve formátu JSON [17] a vypadá následovně:

```
{
 "timestamp": ,  // Casove razitko dat (unix)
 "vlans" : [ // Pole siti
   {
   "tag" : ,        //Unikatni identifikator kazde site
   "privateIPs" : [{                //Pole vnitrnich adres
       "sourceIP" : ,                  //Vnitrni adresa
       "bandwidth" : [{                //Pole komunikaci
                     "targetIP" : ,    //Cilova adresa
                     "download" : {
                                   "tcp" : ,
                                   "udp" : ,
                                   "other" :
                                  },
                     "upload" : {
                                  "tcp" : ,
                                  "udp" : ,
                                  "other" :
                                 }
                     }],
                   }],
      "publicIPs" : [{              //Pole verejnych IP
            "sourceIP" : ,                        //Zdroj
            "bandwidth" : [{          //Pole komunikaci
                  "targetIP" : ,                  //Cil
                  "download" : {
                               "tcp" : ,
                               "udp" : ,
                               "other" :
                                 },
                  "upload" : {
                               "tcp" : ,
```

```
                                    "udp" : ,
                                    "other" :
                                    }
                                }]
                            }]
    }]
}
```

Jelikož data obsahují informace jak o uploadu tak downloadu a ta jsou mezi dvěma adresami identická, pouze s prohozeným označením (download jedné adresy je upload druhé adresy) byla pro odstranění zbytečné duplicity dat zvolena metoda záznamu pouze prvního výskytu této položky komunikace a to k té adrese, která se nachází na seznamu dříve.
V praxi to znamená, že pokud se například v datech vyskytuje komunikace mezi adresami 192.168.0.1 a 192.168.0.5 bude tato zaznamenána pouze u adresy 192.168.0.1, hodnoty pro položky druhé adresy budou poté v programu zrekonstruovány podle potřeby.

Nastavitelné položky pro generaci dat jsou tyto

Mimo minimální a maximální hodnoty zaznamenaných přenosů mezi jednotlivými adresami pro vnitřní i vnější síť lze také nastavit procentuální pokrytí spojů mezi adresami vnitřní sítě (Link coverage).
Vygenerovaný JSON je v prototypu okamžitě použit a je pomocí něj vykreslena příslušná vizualizace. Pokud se uživateli data líbí, může si je také přímo uložit na disk jako soubor ve formátu JSON.

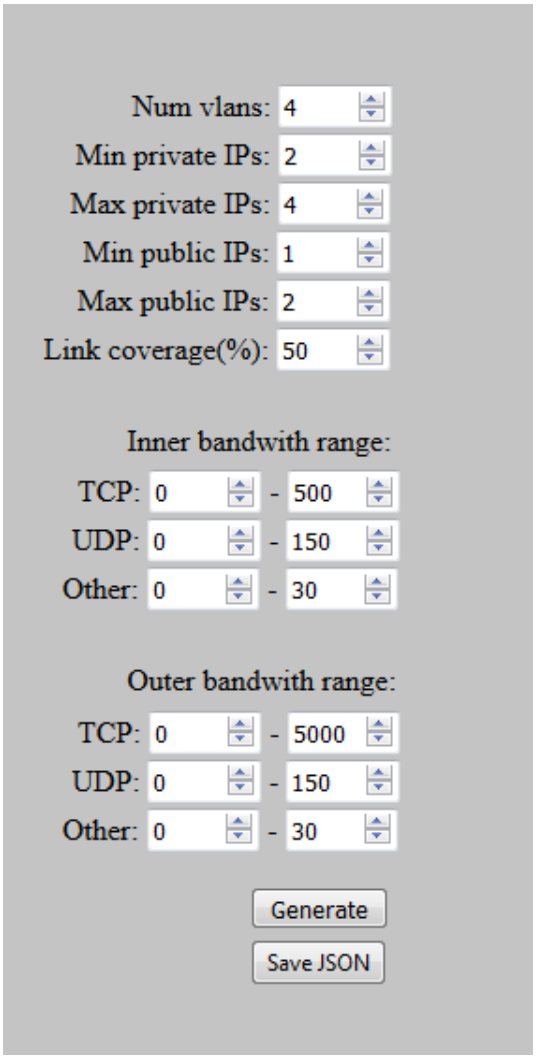
6.4 Technologie prototypu, použití

K implementaci a vykreslení navrhovaných technik byla vytvořena webová aplikace využívající jazyka Javascipt [18] a na něm postavené knihovny D3.js [19] poskytující nástroje pro tvorbu vizualizačních technik.
Nástroj mimo výše popsaného generátoru, jehož použití okamžitě vykreslí nový diagram s vygenerovanými daty také poskytuje čtyři předem vygenerované datasety, které je možné vybrat k zobrazení různých situací

- *Small*
  Malá testovací sada dat, obsahuje 4 sítě o 2 až 4 privátních adresách.

- *Medium*
  Osm sítí v rozsahu 2-4 vnitřní a 1-2 veřejné IP adresy. Vhodné pro základní zhodnocení vizualizace.

- *Big*
  Přibližující se předpokládanému použití, tento dataset vyobrazuje 30



Obrázek 6.5: Generátor dat prototypu

sítí o 2 až 8 privátních adresách s lehce vyšším množstvím existujících spojení.

- *Large VLANS*
  Tento dataset testuje schopnosti techniky vypořádat se se sítěmi obsahující vyšší než očekávané množství IP adres (14-25). Počet sítí je 7 a procentuální pokrytí nastaveno na 30

6.5 Prototyp kruhového diagramu - více sítí

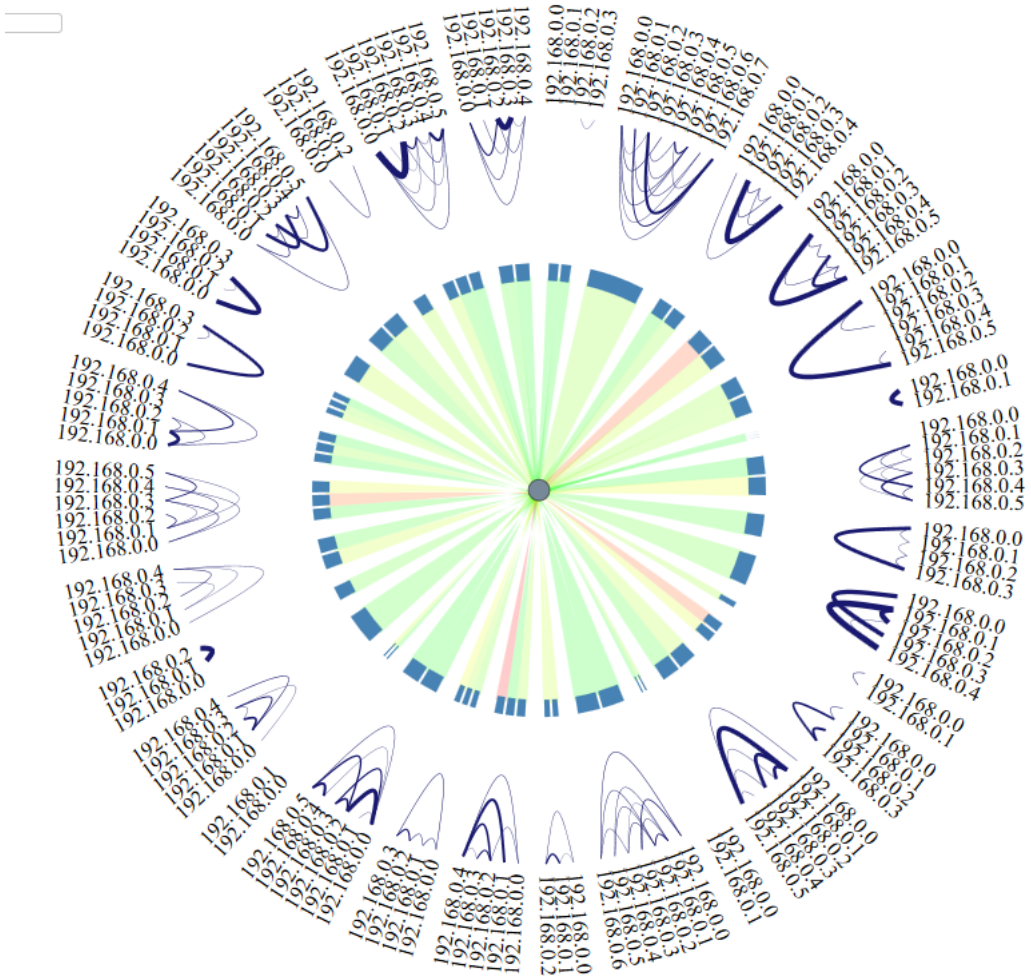
První z naimplementovaných prototypů byla vizualizace přehledu sítí ve formě kruhového diagramu s vyznačením spojů mezi jednotlivými adresami sítí (obrázek 6.6). Příklad vyobrazený na obrázku používá výše popsaný dataset *Big* obsahující 30 sítí o několika adresách. Spoje jsou mezi vytvořeny tak, aby se vzdálenost jejich oblouku od středu kružnice snižovala se zvyšující se vzdáleností dvou adres (čím dále od sebe jsou 2 adresy, tím vyšší oblouk bude) za účelem zvýšení přehlednosti. Jednotlivé linky lze filtrovat podle velikosti za pomocí posuvníku, který v reálném čase odstraňuje linky jejichž hodnoty při svém pohybu překročí (a zase naopak obnovuje ty, které se znovu dostaly do aktivní zóny).
Zobrazení detailů o jedné vybrané síti lze provést kliknutím na jakoukoli IP adresu dané sítě. Dodatečná funkcionalita, která v prototypu není obsažena, ale byla by vhodná při jeho volbě k závěrečné implementaci zahrnuje barevné rozlišení download/upload spojů a snížení viditelnosti těch ostatních k lepšímu zvýraznění příslušných dat při přejezdu myši přes libovolnou IP adresu sítě.
Podobně tak při přejezdu myši po nějaké veřejné IP adrese nebo jejím spoji (struktura uprostřed kružnice) je možné po straně kružnice zobrazit bar chart s aktivitou ostatních veřejných IP adres vůči adrese vybrané, tak jak bylo diskutováno v předešlé části.

6.6 Prototyp kruhového diagramu - jedna síť

Po kliknutí na danou síť je pomocí animace rozevřen její detail, který využívá stejné plochy obrazovky jako celkový přehled. Správné rotace štítků IP adres v prototypu ke zvýšení čitelnosti ještě nejsou řešené, lze tedy momentálně na obrázku (6.7) vidět některé adresy hlavou vzhůru.
Velikost spojů je zde řešena rozsahem vstupních dat, tedy spoj největší velikosti dostane přiřazenou maximální velikost čáry a naopak spoj nejmenší velikosti tu nejmenší. Tato volba by v reálném světě neměla příliš velký informační význam, proto by se tato funkce ve finální verzi řídila uživatelsky nastavenými minimy a maximy, s možností tyto měnit podle aktuální potřeby.

6.7 Prototyp heatmapy - jedna síť

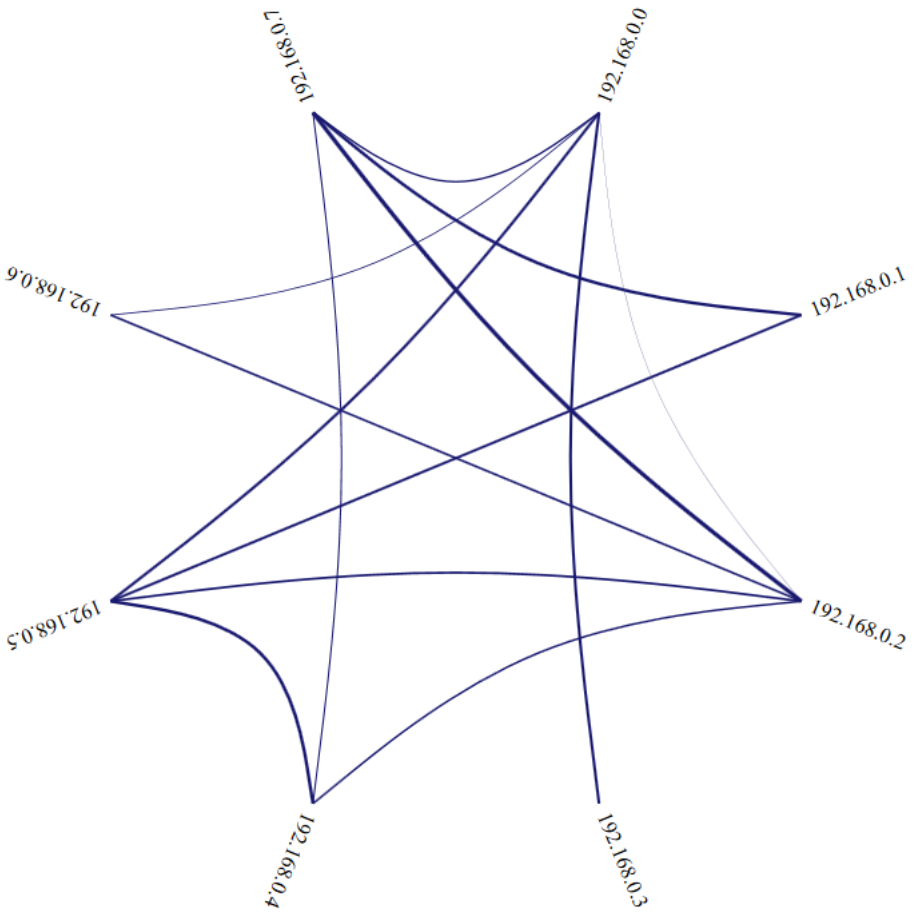
Tato verze nabízí čisté a přehledné informace i pro relativně velké množství vnitřních adres, kde na první pohled můžeme nalézt velmi aktivní nebo naopak neaktivní spoje. Nedochází kde k žádnému křížení informací, pro nezkušeného uživatele však může být obtížnější zjistit jaké hodnoty náleží k jakým adresám. S tímto může pomoci implementace zvýraznění (a ztlumení) příslušných adres/hodnot při přejezdu myší přes relevantní políčka mapy respektive adresy.



Obrázek 6.6: Kruhový diagram sítí

Zobrazení informací pro upload a download případně sumu obou hodnot (v takovém případě bude matice symetrická podle diagonály) lze řešit jednoduchým přepínačem módů (obrázek ukazuje řešení pro download).

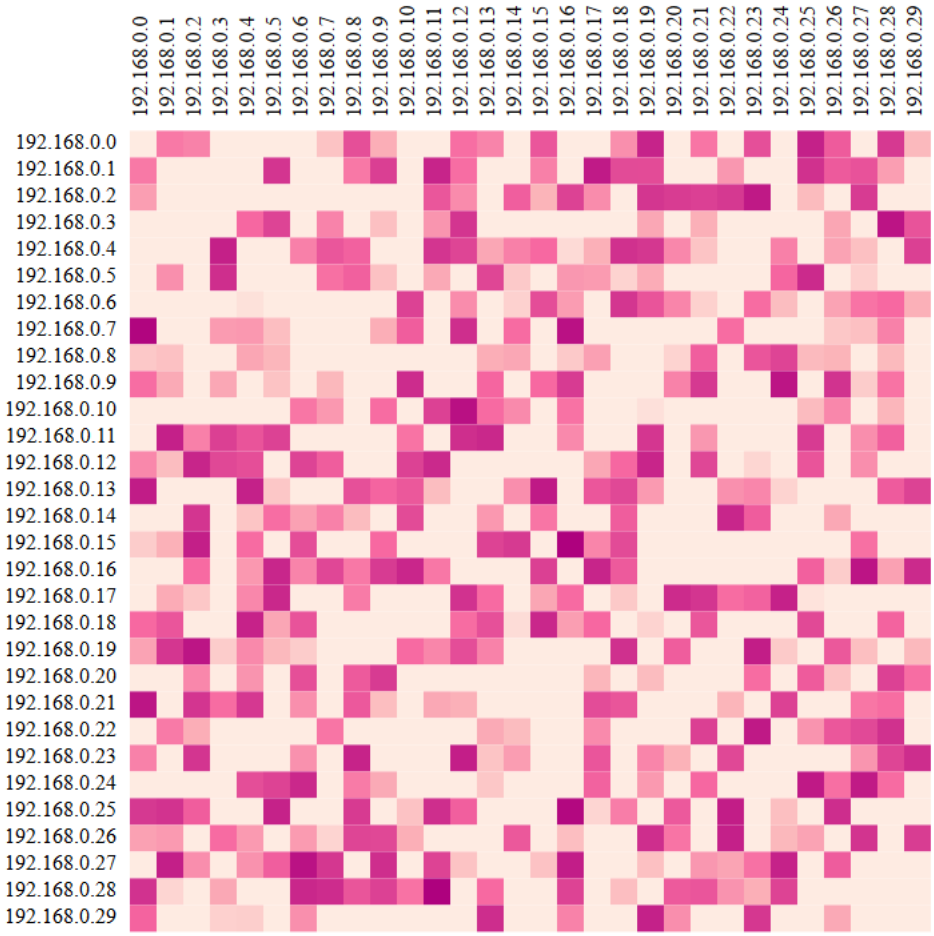
Další existující variantou je současné zobrazení dat pro download i upload podle diagonály (obr. 6.9). Tímto způsobem můžeme využít nabízeného místa k zobrazení dvojnásobného počtu informací na úrovni jednoho diagramu, nevýhodou však je zvýšená obtížnost vyčtení informací z vizualizace. Je v tomto případě totiž potřeba dynamicky měnit směr čtení informací (zleva doprava / ze shora dolů) podle toho co si přejeme zjistit, což se však neřídí žádným pevným pravidlem (například čtení zleva pro upload a čtení ze shora pro



Obrázek 6.7: Kruhový diagram - detail jedné sítě

download) nýbrž pořadím dané adresy (například pro adresu 192.168.0.0 na obrázku zjistíme data u downloadu čtením zleva do prava, pro poslendí adresu 192.168.0.13 potom stejná data čtením ze shora dolů, pro ostatní adresy je toto dokonce smíšené a musí se tedy nalézt správná ze dvou variant).

Mezi hlavní nevýhody heatmapy tedy patří zejména to, že může být velmi náročná na požadované místo na obrazovce a to i v případě malé aktivity spojů, jelikož i neaktivní políčka musí být vždy vykreslena.
Výhodou naopak je škálovatelnost a čistota podávaných informací a možnost zobrazení více typů informací najednou za cenu obtížnějšího čtení.



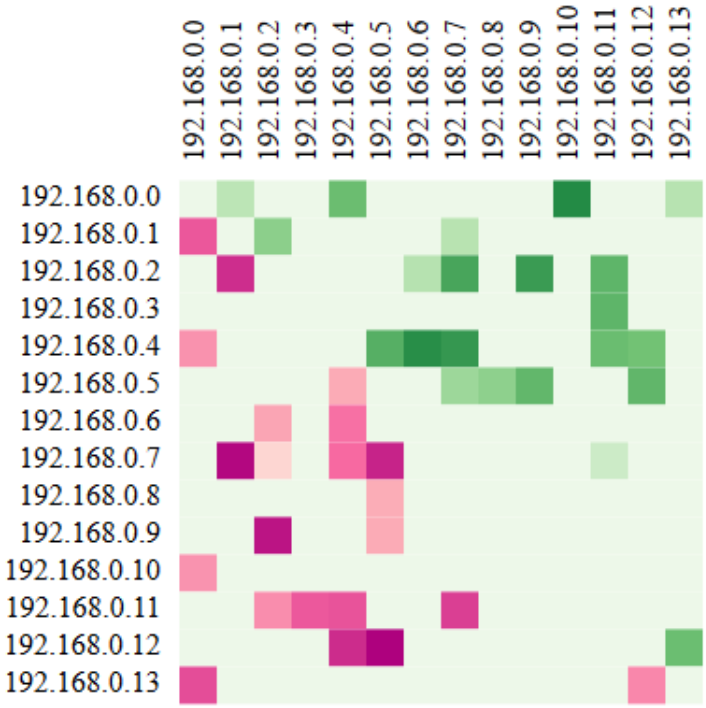
Obrázek 6.8: Prototyp heatmapy pro jednu síť (příklad sítě s 30ti adresami)

6.8 Prototyp heatmapy - více sítí

Problém náročnosti na místo se projevuje právě ve chvíli, kdy potřebujeme vykreslit informace o velkém množství sítí. V takovém případě narážíme na problémy s nutností scrollování obrazovky již po velmi malém počtu sekcí.

Tuto nevýhodu lze částečně vyřešit odstraněním štítků s popisky adres z celkového přehledu, nicméně takový výsledek může postrádat některé údaje nezbytné k vyčtení požadovaných informací přímo z přehledu sítí.

Přestože můžeme tímto způsobem vyobrazit podstatně větší množství sítí, přinejmenším pokud jsou množství jejich vnitřních adres malé, snižuje se hodnota okamžité použitelnosti informace bez nutnosti zobrazení detailu sítě.

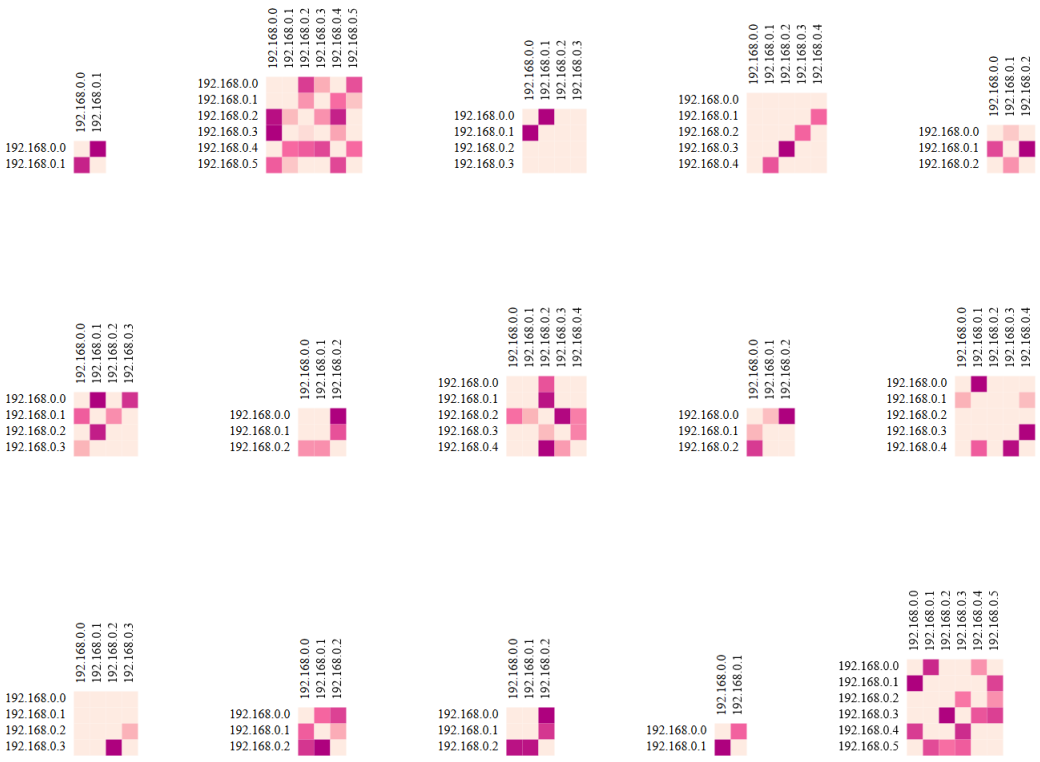


Obrázek 6.9: Ilustrační obrázek zobrazení uploadu (červeně) a downloadu (zeleně) zároveň

Další z možných přístupů by mohl zahrnovat zoomování do velkého pruhu map případně specifické vyhledávání požadované sítě podle označení.

6.9 Shrnutí poznatků, zvolené varianty

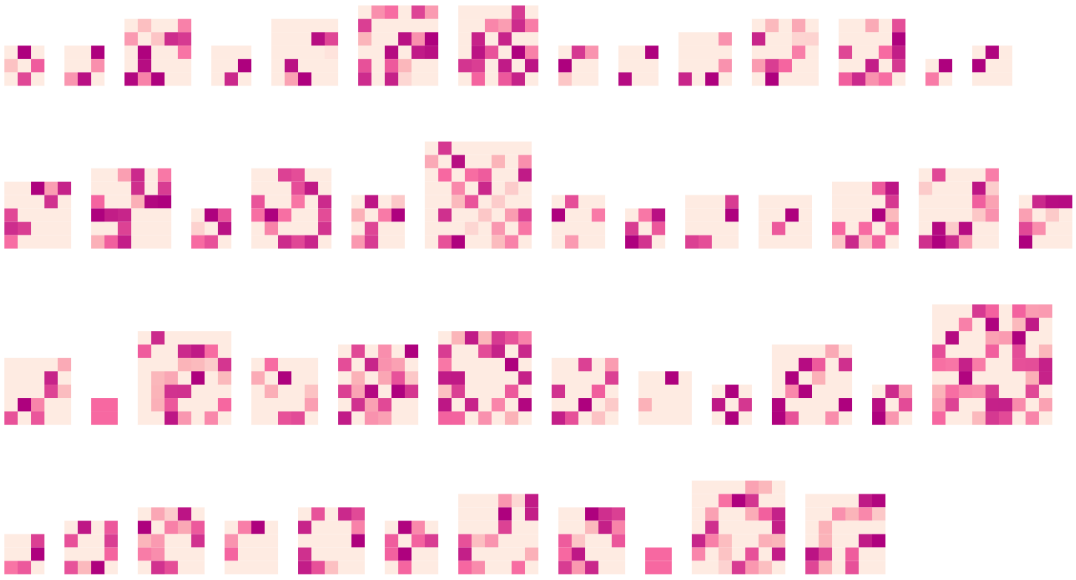
Obě testované metody mají své výhody i nevýhody, lze však říci, že heatmapa nabízí větší škálovatelnost detailního zobrazení informací o jedné síti bez rizika zanesení údajů nečekanou situací, pokud bychom ji však chtěli využít k zobrazení velkého množství sítí, museli bychom se spokojit buď se scrollovacím pruhem obsahující velké množství map variabilní velikosti, s náhodným vyhledáváním pomocí přibližování a oddalování nebo s vykreslením těchto map bez adres popisující jednotlivé řádky a sloupce. Lze zde však argumentovat tím, že v přehledu není nutné vidět detailní popisky jednotlivých řádků a sloupců matice a pouze nám postačuje heatmapu opatřit nějakým identifikátorem, který uživateli dovolí zjistit k jaké síti mapa náleží.
Kruhová vizualizace v současné implementaci bez problémů zvládá zobrazení



Obrázek 6.10: 15 sítí vyobrazených ve formě heatmapy

několika desítek menších sítí a přibližné aktivity v nich a tento rozsah by ještě šlo rozšířit dynamickým nastavováním průměru kružnice podle potřeb, přesto však nakonec narazíme na limitující faktory kdy již nepůjde zobrazit větší množství sítí v jedné kružnici. Další problém zde nastává s detailností samostatného pohledu, kde dochází k výraznému křížení a směsování spojů pro již relativně malé množství adres.

Po detailním zvážení kladů a záporů obou technik a díky velmi líbivým vlastnostem zobrazení detailu sítě za pomocí heatmapy (možnost zobrazení downloadu i uploadu zároveň a vysoká detailnost všech existujících spojů) byla tato metoda zvolena jako způsob pro implementaci detailního náhledu finální aplikace.
Co se zobrazení celkového přehledu týče, pro malé množství sítí v datasetu se kruhová metoda jeví jako velmi zajímavou možností, bohužel zde však dochází k pevné limitaci celkového množství sítí, které je takovýmto způsobem možno zobrazit a tedy podstatnému omezení škálovatelnosti celé aplikace. Z toho důvodu bylo pro zobrazení přehledu sítí také přistoupeno k vizualizační technice heatmapy, v tomto případě však s odstraněnými popisky jednotlivých řádků



Obrázek 6.11: 50 sítí vyobrazených ve formě heatmapy bez štítků

a sloupců (obrázek 6.11), které budou nahrazeny jedním štítkem zobrazující označení dané sítě pro každou z map (tento štítek v referovaném obrázku není vyobrazen).

Kapitola 7

Analýza technologií

V této části práce s již navrženou architekturou aplikace a jejích součástí je čas blíže se podívat na dostupné technologie a vybrat nejlepší možné řešení k implementaci finální aplikace. Při posuzování vhodnosti bude mimo jiné brán zřetel na praktickou zkušenost autora této práce s danou technologií, která je výrazným plusem k úspěšné a efektivní implementaci finálního produktu.
V první části kapitoly stanovíme funkční a nefunkční požadavky aplikace, které nám pomohou s výběrem vhodného technologického zázemí. Na základě těchto bodů vypíšeme seznam možností k použití z nichž nakonec vybereme pro nás nejlepší řešení.

7.1 Součásti aplikace

Jednotlivé části aplikace jsou na sobě relativně nezávislé a je pouze potřeba aby spolu dokázali nějakým způsobem komunikovat, pokud tomu bude nutné. Lze pro ně tedy volit rozdílné technologie.
Aplikace se skládá z následujících součástí:

- *Backend*
  Backendová část produktu se stará o příjímání a poskytování uchovávaných dat, umožňování složitějších operací nad nimi, správu databáze a jiné nezbytné funkcionality. Vzhledem k webové charakteristice frontendu bude podstatná část této funkčnosti vyžadovat formu API, ke kterému bude aplikace schopna přistupovat.
  Toto API bude také použito jako rozhraní na které bude možno sběračem nebo jinou aplikací zasílat námi požadované informace o tocích v sítích, které budou následně zpracovány a uloženy do databáze.
  Do této kategorie také nezbytně patří webový server, který je nezbytný k poskytnutí přístupu jak k námi diskutovanému API, tak k poskytování frontendové části (webových stránek). Některé součásti webové aplikace jsou také typicky prováděny na straně serveru (například kód k zajiš-

tění autentizace a autorizace uživatele). Ve výsledku se tedy potýkáme s následujícími součástmi:

  - API pro práci s daty
  - Serverová část webové aplikace
  - Webový server

- *Databáze*
  Databáze pro uložení zpracovaných dat o tocích a uživatelských nastaveních frontendové části.

- *Frontend*
  Serverem poskytovaná součást vizualizační aplikace běžící ve webových prohlížečích uživatelů. Tato část musí umožňovat dynamickou interakci uživatele s obsahem a také musí být schopna komunikace s API serveru a přizpůsobení zobrazeného obsahu takto získané odpovědi.

7.2 Technologie pro backend

Funkční požadavky

- **Schopnost práce s daty**
  - Načtení dat z databáze
    Aplikace musí být schopna přistoupit do databáze a získat z ní požadovaná data. Pokud možno v co nejmenším funkčním rozsahu (nezískávat data pro jiné než potřebné časové intervaly).
  - Vkládání nových záznamů do databáze
    Zejména v případě příjmu nových dat pro vložení od sběrače je nutné aby tyto byla aplikace schopna správně zpracovat a uložit.
  - Odstranění dat z databáze
    Ať již je tato akce vyvolána pro jeden dataset ručně administrátorem nebo je zadáno odstranění většího množství již nepotřebných dat, je nezbytné aby aplikace poskytovala způsob jakým z databáze odstranit již irelevantní data a nedošlo tak k úplnému zaplnění úložného místa databáze (případně jejímu podstatnému zpomalení kvůli zahlcení).

- **Systém autentizace a autorizace uživatelů**
  Ke komplexnějšímu užití a současnému zajištění bezpečnosti dat je vhodné mít aplikaci chráněnou za autentizačním systémem, který také dokáže indikovat příslušná práva uživatele (například právo mazání dat)

- **Možnost práce se zvoleným typem databáze / databází**

Nefunkční požadavky

- **Serverová aplikace**
- **Operační systém Linux**
- **Stabilita**
- **Bezpečnost**

Zvolené technologie
Jazyků, frameworků a vizualizačních knihoven existuje nepřeberné množství, nebudeme si je zde tedy všechny vypisovat. Namísto toho vyjmenujeme technologie, které jsme pro implementaci této části aplikace zvolili spolu s argumenty proto proč jsme tak učinili.

Nejprve se podívejme na možnosti aplikací webového serveru, kde se nabízí dva hlavní kandidáti - Apache [21] a Nginx [22]. Obě aplikace se pyšní vysokým výkonem a bezpečností a jedná se o běžná řešení používaná k tomuto účelu v produkčním prostředí, je tedy zajištěna jejich stabilita, vysoká bezpečnost a efektivita. Pro námi zamýšlené účely není vyžadována specifická funkcionalita serveru a lze tedy bez problému využít obou z aplikací. V této implementaci bude použito webového serveru Apache.

Samotné HTML stránky jsou kompletně statické, potřebujeme tedy jazyk k zajištění dynamického generování jejich obsahu na straně serveru (server side programming). V tomto případě se nám nabízí dvě hlavní možnosti, jazyky PHP a Ruby.
Ruby nachází své počátky v roce 1995, kdy bylo vytvořeno jako více účelový jazyk. Za své úspěchy v oblasti webového vývoje však vděčí až frameworku Ruby on Rails [23], který vznikl o 10 let později, tedy v roce 2005. Jedná se o jazyk, jehož motto je "Jednu věc lze udělat více způsoby"a používá proto techniky, které nemusí být na první pohled známé ani zkušenému programátorovi. Tato vlastnost dodává nástroji sílu vykonávat komplexní úkoly, je to za to však zaplaceno obtížností naučení se jazyka a debugování stránek takto vytvořených.
PHP bylo vytvořeno ve stejné době jako Ruby, tedy také roku 1995. Narozdíl od Ruby se však jednalo o jazyk od počátku zaměřený na webový vývoj. Tato vlastnost zajišťuje přímočarost použití pro běžné úkoly spojené s vývojem webových stránek, což spolu s podobností jazyka PHP s běžnými programovacími jazyky zaručuje jednoduchost a přímočarost použití.
Žádný z těchto jazyků se nehodí pro tvorbu interaktivního obsahu webových stránek (obsah je vygenerován na straně serveru a poté odeslán uživateli), budeme se proto zaměřovat na jejich použití pro běžné ovládací a administrativní úkony. K těmto činnostem nám bohatě dostačují schopnosti jazyka *PHP*,

které poskytují jednoduchost a účelnost (není tedy potřeba učit se komplexnější technologie Ruby on Rails).

Nyní, když jsme si ustanovili použití jazyka PHP je nutné rozhodnout se, zda-li ho použijeme "čistý"nebo zda-li použijeme některý z existujících frameworků k ulehčení práce. Jednoduchost jazyka a takto napsaného kódu je vítanou vlastností, bohužel se tímto způsobem také setkáme s mnoha bezpečnostními nedostatky, obtížnostmi tvorby některých služeb (zpracování a zabezpečení formulářů, tvorba API atd.) a celkově se vystavujeme nižší robustnosti aplikace.
Pro použití v projektu jsme tedy zvolili PHP framework *Symfony* [24], který je jednou ze v současnosti nejpopulárnějších voleb. Aplikace takto vytvořené nabízí podstatně vyšší možnosti zabezpečení a uživateli jsou poskytnuty nástroje k jednoduchému zajištění běžně požadované funkcionality jako je například přihlášení uživatelů, jejich autentizace a autorizace, použití cookies a mnoho dalšího. Velkou výhodou tohoto prostředí je také inkluze množství nástrojů pro práci s běžnými druhy databází.
Díky předchozím zkušenostem autora práce s tímto frameworkem také odpadá nutnost učení se této technologie od základů.

7.3 Technologie pro frontend

Funkční požadavky

- **Schopnost práce s daty**
  - Získání dat z API
    Je nutné aby aplikace disponovala schopností přístupu k poskytnutému API a získání dat k následnému zobrazení. Tato data je třeba blíže specifikovat k předejití zbytečně velkému přenosu po síti který by obsahoval nepotřebné položky.
  - Aplikace filtrů
    Uživatel by měl být schopný na získaná data aplikovat další filtrační techniky dle aktuálních požadavků.
- **Zobrazení celkového přehledu sítí**
- **Zobrazení detailu jedné sítě**
- **Podpora uživatelů a nastavení**

Nefunkční požadavky

- **Webová aplikace**
- **Dynamická změna obsahu obrazovky (ne statické stránky)**
- **Funkčnost v reálném čase**

Zvolené technologie
Již dříve bylo ustanoveno, že se bude jednat o webové stránky, je tedy zřejmé že pro jejich tvorbu budeme používat jazyků *HTML* a *CSS* kde se nám žádné alternativy nenabízí. Jedná se o časem ověřená a úspěšná řešení.
Nyní je třeba zvolit technologii k zajištění interaktivity stránek v reálném čase bez nutnosti opětovného načtení stránky. Jasnou volbou je zde JavaScript, který je spolu s HTML a CSS jednou ze základních technologií webového vývoje. Zatímco jazyk PHP se stará o zpracování stránky na straně serveru, JavaScript naopak tyto akce provádí v prohlížeči na straně uživatele. Tímto způsobem je proto možné plynule manipulovat s webovou stránkou, změny se však samy o sobě na server nepromítají (pokud je třeba nějaké změny uložit využijeme k tomu backendem poskytovaného API).

I v tomto případě použijeme raději moderní a úspěšné frameworky k ulehčení práce s jazykem a podstatného rozšíření jeho možností. Jedná se zejména o framework *D3.js*, vyvinutý za účelem tvorby vizualizačních webových stránek, který je v dnešní době velmi populární. Díky své architektuře postavené na manipulaci DOM prvků HTML nabízí téměř neomezené možnosti, nevýhodou je složitější použití než-li u běžných jednoúčelových nástrojů.

Dále v práci použijeme oblíbený JavaScript framework *JQuery* [25], poskytující nástroje pro efektivnější a přehlednější psaní kódu a také jeho rozšíření *JQuery UI*, které nabízí vylepšené prvky uživatelského rozhraní s dodatečnou funkcionalitou.

7.4 Volba databáze

Předpokládaný charakter dat
U nasbíraných dat se předpokládá charakter shlukování do více či méně velkých skupin. Pod tímto si lze představit například velké "balíčky"označené časovým razítkem, každý z nichž obsahuje další balíky dat pro jednotlivé zdrojové IP adresy, použité protokoly atd.
Ke zpracování a vizualizaci si tedy z databáze budeme žádat zejména velmi objemné balíky dat založené na nějaké jejich sdílené charakteristice (např. již zmíněné časové razítko).

Možné varianty

- Relační SQL databáze (MySQL [26] / PosgreSQL [27])
  V praxi velmi populární volba, v současné době však již pro určité případy existují výkonnější alternativy. Podstata fungování těchto databází je založená na rozdělení dat do mnoha skupin specifických podob dodržující normalizační pravidla. Tímto je zajištěn výkon při dotazech, nicméně data jsou fragmentována a celé tabulky s jejími přináležícími prvky musí být při každém dotazu znovu sestaveny.
  Nejedná se tedy o velmi vhodné řešení v případech, kdy pracujeme s velkými skupinami na sobě závislých dat, což je náš předpokládaný případ. Volba tohoto typu databáze tedy není vhodným řešením.

- Wide Column Store (Apache Cassandra [28])
  Tento typ databáze padá do moderní tzv. noSQL kategorie. Využívá tabulky, jejichž formát je nutné definovat, ale na rozdíl od klasické SQL tabulky je možné s ním libovolně manipulovat . Data jsou uložené po řádcích, kde každý sloupec v řádku představuje jeden ze zadaných datových typů (string, list, set, mapa...) a daný sloupec je možné při vkládání řádku také úplně vynechat (v tom případě bude jeho hodnota nastavena na NULL). Tabulka tedy představuje kolekce řádků, které sdílejí stejné nebo podobné sloupce.
  Dotazovací jazyk této databáze (CQL - Cassandra Query Language) poskytuje základní vyhledávací funkce, jeho filtrovací a podmiňovací schopnosti jsou však podstatně omezenější než u klasického SQL.

- Big Data (Apacha Hadoop [29])
  Další z kategorie noSQL databází, tentokrát ze skupiny Big Data. Tento typ databází se používá v případě, že objem a rychlost růstu dat se kterými pracujeme je podstatně větší (exponenciálně) než u běžných případů. Databáze sestává z množství serverů na bázi master-slave, distribuovaným souborovým systémem a schopnosti paralelního zpracování komplexních operací nad daty za pomocí MapReduce.
  Přestože se jedná o výkonné řešení, jeho rozsáhlost (desítky, stovky nebo i tisíce dedikovaných serverů) a komplexnost dalece přesahuje požadavky databáze pro naši vizualizační aplikaci (není žádané aby systém vizualizace byl stejné velikosti jako cloud, který zobrazuje). Díky předpokladu agregace dat a tedy podstatné redukce jejich velikosti se tedy nejedná o vhodné řešení.

- Document Database (MongoDB [30])
  Finální z noSQL databází, které si zde představíme spadá do kategorie dokumentových databází. Data jsou v tomto případě uložena v kolekcích souborů, kde za soubory jsou považovány data ve formátu JSON

(v db uloženy binárně jako BSON). Velmi vlídná nastavitelnost a hiearchie tohoto datového formátu dovoluje ukládání rozmanitého množství informací bez nutnosti dodržovat předem definované formáty tabulek. Databáze umožňuje zadávání komplexních dotazů a filtrování za pomocí jazyka JavaScript týkajících se jak identifikátorů tak obsahů uložených dokumentů. Pro komplexnější případy lze také využít rozhraní poskytující schopnosti MapReduce pro paralelní zpracování dat.

Zvolená databáze
Vzhledem ke skupinové podstatě formátu dat, nevhodného ke zpracování běžnou SQL databází a jejich velikosti nevyžadující vstupu do oblasti Big Data, nám z vypsaných možností zbývají pouze dvě volby a to databáze Apache Cassandra a MongoDB. Cassandra poskytuje flexibilitu ukládaných dat a jejich seskupení do řádků tabulky, požadovaná hiearchie našich dat však může zacházet do více úrovní, kde tato varianta přestává být vhodná. Omezené dotazovací schopnosti poté eliminují možnosti filtrování dat již při jejich vyhledávání v databázi.
Finální volbou se tedy nakonec stává databáze MongoDB, která poskytuje možnosti ukládání rozlišných a hiearchicky komplexních skupin dat ve formátu JSON. Tento formát a dotazovací jazyk JavaScript také zaručují vysokou kompatibilitu s frontendovou částí aplikaci, postavené na webové platformě, která využívá tento jazyk.

Po boku této aplikace také využijeme databázi MySQL k ověřování uživatelů a načítání a ukládání jejich nastavení. Použití této databáze by nebylo nezbytně nutné, nicméně tímto způsobem nebudeme při velkém vytížení databáze s hlavními daty zpomalovat pokusy uživatelů o přihlášení do aplikace.

7.5 Možné technologie pro sběrač dat

Funkční požadavky

- **Schopnost práce s daty**
  - Přenos dat od sběru k backendové části
    Nasbíraná data je nutné přenést ze zdroje na API nacházející se na backendové části aplikace. Toto je možné udělat vlastními prostředky (napsání TCP/UDP rozhraní a odesílací/přijímací část aplikace) nebo externími nástroji (existující linuxové programy).
  - Zpracování a agregace dat
    Data je před uložením nutné zpracovat do vhodného formátu reprezentující jejich relace a časovou složku jejich záznamu. Součástí zpracování je také agregace položek k zachování rozumné velikosti databáze. Tyto části lze provést přímo sběracím nástrojem aplikaci nebo za pomocí existujících nástrojů třetí strany.

  - Čtení dat z API
    Přestože to není součástí navrhované základní funkcionality, v určité chvíli si vývojář může přát načíst nějaké informace z databáze (vizualizační nebo cloudové) pro provedení porovnání a dalších akcí nezbytných ke komplexnějším funkcím.

Nefunkční požadavky

- **Funkčnost v linuxovém prostředí**
- **Možnost deamonizace či opakovaného spuštění**
- **Využití existujících nástrojů a protokolů, tam kde je to vhodné**

Možnosti
Přestože implementace sběrače není součástí této práce a při splnění výše popsaných nezbytností je použitá technologie libovolná, zde je výpis některých vhodných možností pro budoucí implementaci.

- C/C++
  Oblíbená volba pro nespočet aplikací za posledních několik desetiletí. Tento jazyk spadá do vyšší třídy programovacích jazyků založených na objektovém programování (C++) zatímco si zachovává nástroje pro více hardwarově orientované operace (práce s pamětí či procesorem). Mezi hlavní výhody patří kompatibilita mezi platformami, vysoká granularita a přehlednost kódu a dobrá schopnost práce se systémovými nástroji.
  Za negativum můžeme považovat větší rozsáhlost kódu a větší pracnosti k provedení některých operací (to lze však z velké části eliminovat využitím některé z mnoha knihoven, které se na internetu nacházejí, kde jsou tyto často vyhledávané funkce již implementovány).

- Java
  Další jazyk spadající do kategorie programovacích jazyků vyšší třídy. Je objektově orientovaný a své funkčnosti dosahuje zejména využitím velkého množství tříd obsahující nezbytné operace. Tímto se odlišuje od již zmíněného jazyka C++, tím že uživateli nabízí možnost zabývat se cílem operace a ne jejími implementačními detaily. Nevýhodou naopak je větší nutnost znalostí hiearchie tříd, jejich metod a dalších detailů.
  Jazyk funguje na množství platforem tím způsobem, že ke svému běhu využívá virtuálního Java prostředí (Java Virtual Machine), které je nutné na každý systém kde si program přejeme spustit nejprve nainstalovat.

- Python
  Jedná se o vysokoúrovňový skriptovací jazyk. Svou funkčností nabízí podobné možnosti jako v případech jazyků C či Java, hlavním rozdílem však je skriptová povaha kódu. Programy napsané tímto jazykem jsou vnímány jako skripty, tedy se spustí, vykonají definovanou akci a zase

se ukončí. Pro účely zpracování a uložení dat se jedná o vhodnou volbu, jelikož je tato akce prováděna pouze jedenkrát za každý časový interval a není tedy nutné aby aplikace v době své neaktivity běžela na pozadí.

7.6 Shrnutí

Závěrem kapitoly si shrňme, které technologie a jazyky jsme se rozhodli pro implementaci jednotlivých částí aplikace používat. Důvody pro jejich zvolení jsou popsané v jejich dedikovaných sekcích výše.

- Backend
  - PHP
  - Symfony PHP framework
- Frontend
  - HTML
  - CSS
  - JavaScript
  - D3.js
  - JQuery
  - JQuery UI
- Databáze
  - MongoDB
  - MySQL

Kapitola 8

Návrh backendu

V této části si blíže specifikujeme návrh backendové části aplikace a to zejména jejího API a dalších relevantních částí jako je tvar databází a způsobů jakým budou použity. Přestože jsou programy a části jako webový server, databáze atd také součástí této kategorie, není v tomto ohledu potřeba nic navrhovat a detaily jejich konfigurace tedy ponecháme až na část implementace samotné.

8.1 Popis zvoleného PHP frameworku Symfony a nezbytných součástí

PHP Symfony je framework určený k rychlé a efektivní tvorbě webových stránek a aplikací. Poskytuje rozhraní využívající desítky populární a na sobě nezávislých nástrojů pro moderní vývoj webových aplikací a slučuje je do jednoho mocného nástroje upravitelného podle aktuálních potřeb uživatele, všechny tyto nástroje totiž lze libovolně přidávat a odstraňovat ve formě tzv. balíků (bundles). Dokonce i aplikace takto vyvíjená je také součástí svého vlastního bundlu, který je do frameworku připojen. Pro započetí návrhu aplikace v tomto frameworku si nejprve musíme vysvětlit jeho hlavní součásti a princip funkcionality. Podívejme se tedy na adresářovou strukturu projektu a popišme si pro nás zajímavé komponenty, které bude třeba modifikovat či rozšířit.

- app
  Adresář obsahující nastavení aplikace, lze zde nalézt konfigurační soubory pro systémy databází, zabezpečení, směrování a jiné. Podsložka Resources také obsahuje obecnou rozšiřitelnou šablonu naší webové stránky (která je nadále rozšířena podšablonou ve složce zdroje aplikace)

- bin
  Zde se nachází ovládací konzole frameworku, o použití více v implementaci

- src
  Zdrojová složka obsahující bundle naší aplikace. V této části lze nalézt veškerý aplikaci specifický PHP kód jako jsou controllery, šablony, databázové entity atd.

- web
  Složka obsahující veškerý webový obsah, který má být uživateli přístupný (například obrázky, javascript, css atd.)

- composer.json
  Symfony pro jednoduchost zajištění požadované konfigurace používá php nástroj composer. V tomto souboru lze specifikovat požadované balíky a nástroje, které pro nás poté composer (po spuštění patřičného update příkazu z konzole) obstará a zakomponuje do frameworku.

Z požadavků na aplikaci a schopnostem frameworku bude pro implementaci nutné spolu k základní funkcionalitě balíky pro následující funkce:

- REST API
  Přídavek poskytující frameworku funkčnost tvorby a komunikace s REST API využívající volání url adresy se specifikací HTML metody (jako jsou GET, POST, PUT, DELETE)

- JSON serializer
  Pro komunikaci prostřednictvím API je nezbytné aby aplikace dokázala převádět objekty z PHP tříd do formátu JSON a naopak

- MongoDB driver
  Pro zajištění schopnosti práce s námi zvolenou databází pro uchovávání vizualizačních dat (ovladač MySQL databáze pro obsluhu dat uživatelů je již součástí základních schopností frameworku)

8.1.1 Proces zpracování požadavku

Ve chvíli kdy je na aplikaci zaslán URL požadavek o nějakou činnost, reakce aplikace je v obecném případě zhruba následující:

URL adresa je předána směrovači, který rozhodne jak s ní naložit. Symfony nabízí množství způsobů jak směrování nakonfigurovat (anotace, soubory yml, PHP). V našem případě budeme používat konfiguraci pomocí souborů .yml z důvodu osobních preferencí (všechny způsoby konfigurace jsou rovnocenné). Tyto soubory se v aplikaci nacházejí 2, jeden pro globální směrování (spuštění správného balíku) nacházející se v adresáři "app/config/routing.yml"- tento soubor obsahuje pouze instrukce ke zvolení správné aplikace.

Druhý konfigurační soubor se nachází již v kódu aplikace samotné "src/jménoBalíkuAplikace/Resources/config/routing.yml"a obsahuje odkazy na specifické controllery podle tvaru URL adresy.

Po zpracování směrovačem je zavolán příslušný controller. Jedná se o PHP třídy a metody (vždy je volána specifická metoda specifické třídy) nalezené ve složce "src/jménoBalíkuAplikace/Controllers". Tento PHP kód provede zde specifikovanou akci (například zpracování parametru URL, načtení dat z databáze a jejich případné zpracování atd.) a zavolá specifickou HTML šablonu poskytnutou šablonovacím systémem Twig [32], do které může předat zde získaná data.

Posledním krokem zpracování je vygenerování zavolané šablony, které může obsahovat akce jako vyplnění z PHP poskytnutých parametrů atd. Tyto šablony lze nalézt ve složce "src/jménoBalíkuAplikace/Resources/views".
Takto vytvořená HTML stránka je poté finálně navrácena uživateli jako odpověď na přijatý dotaz.

8.1.2 Komponenty procesu

Na základě tohoto popisu zpracování požadavku je tedy zřejmé že v implementaci budeme mimo jiné muset vytvořit následující komponenty procesu:

- Router
- Controller
- HTML template

Samozřejmě jak již bylo řečeno, námi navrhovaná aplikace se bude chovat především jako jedna webová stránka využívající REST API pro komunikaci se serverem, zpracování takovýchto požadavků bude však téměř identické jako tento proces, rozdíl se bude nacházet pouze v poslední části, kdy namísto navrácení HTML šablony z controlleru budou navrácena serializovaná data ve formátu JSON.

8.1.3 Práce s databází

S každou z obou zamýšlených databází bude z frameworku komunikováno především z jedné z těchto popsaných komponent a to z controllerů. Tyto PHP třídy ve svém kódu obstarávají práci s databázemi a k ulehčení práce používají nástroj Doctrine [31].
Tato knihovna zařizuje značnou abstrakci od běžné práce s databází a dovoluje načítání položek databáze do klasických PHP tříd, se kterými lze v kódu pohodlně pracovat (a naopak převod dat z PHP třídy do databáze).

Doctrine (po nainstalování potřebného rozšíření) dokáže pracovat jak s databází MySQL, tak s MongoDB při zachování velmi podobného procesu úkonů. Nejpodstatnější rozdíl mezi těmito dvěma verzemi se nachází zejména v definici PHP tříd, které budou data držet.

Doctrine pro obsluhu MySQL používá takzvané entity (nalezené v adresáři "src/jménoBalíkuAplikace/Entity"). Jedná se o běžně definovanou PHP třídu, jejíž proměnné jsou obohaceny komentářům podobnými anotacemi vyjadřující jejich vztah vůči databázi a ostatním položkám. Jednoduše řečeno jedna takováto entita v podstatě vyjadřuje jednu tabulku databáze (a jedna specifická instance takového třídy obsahuje 1 řádek této tabulky).
Definicí všech entit a jejich anotací vytvoříme schéma databáze, pro které poté (za pomocí konzole) Doctrine vygeneruje patřičné SQL příkazy a vytvoří naši databázi. Během celého procesu tedy není nutné jakékoli použití SQL či přímé interakce s databází.

Verze Doctrine pro MongoDB funguje ve svém principu velmi podobně, namísto entit však používá definice dokumentů (adresář "src/jménoBalíkuAplikace/Document"). Opět se jedná o PHP třídy obohacené anotací, princip těchto anotací se však mírně liší z důvodu odlišnosti architektury databáze. Jelikož dokumenty ukládané v této databázi jsou JSON soubory, které mohou obsahovat další vnořené JSON podsoubory a pole je třeba tyto třídy navrhnout takovým způsobem aby dokázaly obsáhnout celý zamýšlený soubor JSON (tj. správné navrhnutí hierarchie tříd).

8.1.4 Návrh entit pro MySQL

Jelikož je tato aplikace použita pro správu dat uživatelů, musí obsahovat třídy je referující. Zde popsané třídy (user a role) jsou navrženy za účelem kompatibility s implicitním autentizačním a autorizačním systémem frameworku a mohou se v implementaci tedy mírně lišit od běžné entity (zejména děděním odlišného interface třídy).
Pro nás nezbytné třídy a jejich proměnné jsou následující:

- User - tabulka uživatelů
  - id - identifikátor uživatele (integer)
  - username - jméno uživatele (string)
  - password - hash hesla (string)
  - email - emailová adresa uživatele (string)
  - apiKey - identifikátor uživatele pro potřeby API (string)
  - isActive - označuje povolení přihlášení do systému (boolean)
  - roles - role uživatele (array[Role])

  - preferences - třída obsahující nastavení uživatele (Preferences)

- Role - tabulka rolí (user, administrator atd.)
  - id - identifikátor role (integer)
  - name - jméno role (string)
  - role - označení role pro účely Symfony (string)
  - users - uživatelé mapované na roli (User)
- Preferences - tabulka nastavení uživatele, obsah dle aktuálních potřeb

8.1.5 Návrh dokumentů MongoDB

Zopakujme si zde nejprve námi navrhovaný formát souboru JSON pro uchování dat v MongoDB, již dříve vyobrazený v kapitole 6.

```
{
 "timestamp": ,  // Casove razitko dat (unix)
 "vlans" : [ // Pole siti
   {
   "tag" : ,       //Unikatni identifikator kazde site
   "privateIPs" : [{               //Pole vnitrnich adres
       "sourceIP" : ,                 //Vnitrni adresa
       "bandwidth" : [{              //Pole komunikaci
                      "targetIP" : ,    //Cilova adresa
                      "download" : {
                                    "tcp" : ,
                                    "udp" : ,
                                    "other" :
                                   },
                      "upload" : {
                                   "tcp" : ,
                                   "udp" : ,
                                   "other" :
                                  }
                     }],
                   }],
      "publicIPs" : [{             //Pole verejnych IP
            "sourceIP" : ,                        //Zdroj
            "bandwidth" : [{          //Pole komunikaci
                  "targetIP" : ,                  //Cil
                  "download" : {
                                "tcp" : ,
                                "udp" : ,
                                "other" :
```

```
                        },
                "upload" : {
                        "tcp" : ,
                        "udp" : ,
                        "other" :
                        }
                    }]
                }]
    }]
}
```

Nyní je třeba za pomocí anotací navrhnout PHP třídy, které tento formát souboru dokáží obsáhnout. Doctrine nám v tomto případě dovoluje dvou způsobů návrhu, jeden ukládající data v databázi pomocí referencí na jednotlivé součásti a druhý přímým obsáhnutím obsahu podtřídy v dokumentu. V našem návrhu využijeme postup druhý, tedy budeme ukládat celý dokument jako celek v jediné kolekci.
Návrh hierarchie tříd je následující:

- NetworkData - třída obsahující celý soubor
  - timestamp - časové razítko použité jako identifikátor dokumentu (string)
  - vlans - kolekce všech dat virtuálních sítí pro daný dataset (array[Vlan])
- Vlan - Data pro jednu virtuální síť v daném data setu
  - tag - označení sítě (integer)
  - privateIPs - data toků vnitřních IP (array[PrivateIP])
  - publicIPs - data toků veřejných IP (array[PublicIP])
- PrivateIP - Záznam pro jednu adresu
  - sourceIP - identifikátor zdrojové adresy (string)
  - bandwidth - datový tok vůči ostatním adresám (array[Bandwidth])
- Bandwidth - Jeden záznam toků pro jednu cílovou adresu
  - targetIP - identifikátor cílové adresy (string)
  - download - data o stahování (array[Protocols])
  - upload - data o uploadu (array[Protocols])

- Protocols - informace o toku u jednotlivých protokolů
  - tcp - velikost toku využívající TCP (integer)
  - udp - velikost toku využívající UDP (integer)
  - other - velikost toku využívající jiný protokol (integer)
- PrivateIP - záznam pro jednu adresu
  - sourceIP - identifikátor veřejné adresy (integer)
  - bandwidth - agregovaná suma toků podle protokolů (array[Protocols])

8.2 Návrh API

V této části se zaměříme na návrh funkčnosti API, nezbytných volání k běhu aplikace a přiřazení těchto volání do controllerů, které je budou obsahovat. Součástí návrhu bude také zamýšlená url adresa volání (relativní vůči domovské stránce aplikace, tzn. adresa ve stylu "/api/call"bude referovat adresu "www.stránka-aplikace.com/api/call") a použitá HTML metoda.
Ačkoli rozšiřitelné, počáteční podoba API bude sloužit ke dvěma hlavním účelům - načítání a správa dat uživatelů a načítání a správa vizualizačních dat.

8.2.1 Správa uživatelů

- Získání dat uživatele (url: /api/user, metoda: GET)
  Tato funkce navrátí informace o aktuálně přihlášeném uživateli (v prohlížeči ze kterého je volána). Použití je zamýšleno primárně na získání informací o uživateli v rozhraní frontentu.
  Controller: Api/UserController
- Získání uživatelova nastavení (url: /api/user/preferences, metoda: GET)
  Výstupem volání je JSON obsahující informace o všech poskytovaných uživatelských nastaveních.
  Controller: Api/UserController
- Uložení uživatelova nastavení (url: /api/user/preferences, metoda: POST)
  Na zadanou url adresu je zaslán JSON soubor obsahující současné volby nastavení z frontendu aplikace, tyto jsou uloženy do databáze.
  Controller: Api/UserController

8.2.2 Správa vizualizačních dat

- Získání všech data setů v databázi (url: /api/data, metoda: GET)
  Volání vrátí kolekci všech data setů momentálně uložených v databázi. Jelikož se jedná o potenciálně velmi obsáhlé volání je toto určeno především pro vývojové účely.
  Controller: Api/NetworkDataController

- Načtení seznamu časových razítek všech data setů v databázi (url: /api/data/timestamps, metoda: GET)
  Tímto voláním získáme JSON pole obsahující seznam všech časových razítek nacházejících se v databázi (1 razítko = 1 data set) seřazených vzestupně. Tato funkce bude využita k přečtení současného obsahu databáze při výběru dat k načtení.
  Controller: Api/NetworkDataController

- Získání všech data setů v databázi (url: /api/data/timestamp, metoda: GET)
  Navrátí jeden specifický data set podle parametru timestamp obsaženého ve volání (bez). Pokud je výsledek prázdný vrátí kód 204 (no content).
  Controller: Api/NetworkDataController

- Uložení validního data setu do databáze (url: /api/data, metoda: POST)
  Zaslání souboru JSON ve výše specifikovaném formátu na tuto adresu docílíme uložením dat do MongoDB a možnost jejich následného načtení ve frontendu aplikace. Pokud jsou na rozhraní zaslána data obsahující časové razítko souboru v databázi již uloženém je tento soubor nahrazen.
  Controller: Api/NetworkDataController

- Smazání specifického data setu (url: /api/data/timestamp, metoda: DELETE)
  Tato funkce slouží k odstranění specifického data setu z MongoDB identifikovaného jeho časovým razítkem zaslaným jako parametr volání.
  Controller: Api/NetworkDataController

8.3 Použitá HTML šablona

Aplikace bude obsahovat dvě HTML šablony.
První z nich bude poskytovat přihlašovací obrazovku, která se bude nacházet na samostatné stránce. Tato šablona je využívána controllerem starající se o zabezpečení aplikace (""src/jménoBalíkuAplikace/Controller/SecurityController") a bude obsahovat přihlašovací formulář.

Druhá a hlavní šablona bude reprezentovat domovskou stránku aplikace. Budou se v ní tedy nacházet odkazy na javascriptové, css a jiné soubory obsažené ve složce "web", které za pomoci výše definovaných API volání budou zajišťovat dynamickou funkčnost aplikace. Pro dodržení konvencí bude tato šablona rozšiřovat základní kostru HTML stránky obsahující všechny potřebné headery nacházející se v "app/Resources/views/base.html.twig. Stránka bude poskytována jednoduchým controllerem zvaným IndexController obsaženým ve složce Controllers , který bude vyvolán při vložení domovské adresy aplikace a navrátí tuto stránku.

8.4 Shrnutí návrhu

V této kapitole jsme si popsali základní principy frameworku, který bude k implementaci použit a definovali části, které bude třeba vytvořit či modifikovat.
Také jsme si definovali obsahy obou databází a REST API rozhraní, které nám k nim bude poskytovat přístup a v jakých souborech se tyto kódy budou nacházet.
Nakonec jsme si ujasnili jaké HTML šablony se v aplikaci budou vyskytovat, co je bude volat a co budou obsahovat.

Kapitola 9

Návrh frontendu

Tato kapitola se bude zabývat návrhem funkcionality frontendové části aplikace a to především za pomoci volání v předchozí kapitole definovaného API.

9.1 Zvolení architektury webové aplikace

Dle typu použitých technologií a vhodnosti pro způsob použití lze nalézt různé typy webových aplikací, jejichž hlavní rozdělení spočívá zejména v tom, zda-li je aplikace rozdělená na více stránek, které se musí jednotlivě načítat a nebo zda-li aplikace používá formu jedné webové stránky, ve které jsou všechny změny promítnuty dynamicky.

První ze způsobů má své výhody v jednodušším způsobu implementace, jelikož lze v tomto případě velké množství prvků stránky vygenerovat již na straně serveru v okamžiku zadání požadavku a ty jsou poté přeneseny do uživatelova prohlížeče. Také je tímto způsobem limitováno množství paměti, které musí prohlížeč v jednu chvíli uchovávat k uložení všech dat, které aplikace v danou chvíli využívá.
Nevýhodou tohoto přístupu však je méně či více častá nezbytnost znovu načtení celé stránky, což mimo jiné zahrnuje ztrátu současně uložených informací a především působí jako rušivý element na pozornost a plynulost práce uživatele.

Druhým možným přístupem, které množství moderních aplikací využívá je vzdálení se od klasického konceptu množství nezávislých webových stránek a koncentrace celé (či funkční většiny) aplikace do stránky jediné. Tato varianta vyžaduje využití moderních technik k zajištění dynamické reaktivity celé stránky a přizpůsobování obsahu aktuálním požadavkům uživatele. Je tedy nutné nějakým způsobem v pozadí asynchronně komunikovat se serverem a průběžně obsah aplikace překreslovat.
Využití tohoto způsobu dovoluje dosažení vysoké plynulosti práce s aplikací

a minimalizaci distrakcí uživatele, může však být nezbytné udržovat v paměti počítače větší množství informací a výpočetní proces generování stránky se také přesouvá do prohlížeče uživatele namísto jeho provedení na serveru. Tento typ architektury nemusí také vyhovovat všem typům použití.
Jelikož však naše aplikace vyžaduje interaktivitu s uživatelem a navíc se jedná o vizualizační nástroj, ve kterém je žádané minimalizovat veškeré vyrušující faktory ovlivňující paměť a koncentraci uživatele, byla pro použití v implementaci zvolena architektura jedná dynamické webové stránky.

9.2 Stavy aplikace

V této části máme ujasněno, že budeme pracovat s jednou dynamicky překreslující se webovou stránkou, je však ještě třeba blíže specifikovat jednotlivé stavy, ve kterých aplikace může být, které odlišují její chování vůči akcím uživatele.

9.2.1 Přehled sítí

Výchozí stav aplikace, ve kterém stránka zobrazuje celkový přehled aktuálně načtených sítí (případně žádné sítě a možnost načtení data setu pokud ještě nebyl žádný zvolen). V tomto módu si může uživatel prohlížet všechny v data setu obsažené sítě a jejich zatížení relativní vůči sobě (tzn. barevná stupnice bude volena s globálním maximem jako nejvyšší hodnotou ze všech načtených sítí). Z této obrazovky bude kliknutím na libovolnou ze sítí možné přejít do režimu detailu.

9.2.2 Detail sítě

Po výběru sítě jsou všechny ostatní z obrazovky odstraněny a vybraná síť je obohacena o popisky a interaktivní funkcionalitu popsanou dále. Tento režim bude obsahovat tlačítko, kterým sel ze vrátit zpět do přehledu, barevnou škálu která je pro vizualizaci použitá a bar chart pro zobrazení zátěže veřejných IP adres k této síti náležících.

9.3 Funkční součásti

Každý z námi specifikovaných módů bude mít níže specifikovanou funkcionalitu.

9.3.1 Vlastnosti přehledu

Samozřejmou funkční vlastností kterou budeme od přehledu všech sítí vyžadovat je již dříve specifikované zobrazení pole heatmap se štítkem označující tag virtuální sítě ke které tato mapa náleží.

Mimo jiné z této obrazovky také musí být možné načíst data, která si na ní přejeme zobrazit a to pokud možno nějakým přehledným způsobem (výběr dne, hodiny atd). Po takto provedeném výběru bude předchozí obrazovka přehledu odstraněna a překreslena novou takto načtenou.

K vyšší přehlednosti všech načtených dat, zejména v případě jejich velkého objemu bude sloužit nalevo ukotvený seznam obsahující položky pro všechny sítě a indikátor velikosti zatížení sítě vůči těm ostatním. Také by zde mělo být možné zobrazit IP adresy, které k síti patří (po kliknutí) a přejít přes tuto položku přímo do jejího detailního zobrazení.
Položky tohoto seznamu budou obsahovat dvě hlavní hodnoty - identifikátor a informace o zatížení sítě. Je žádané aby tento seznam (a korespondující heatmapy v přehledu) bylo možné seřadit jak podle označení, tak podle zátěže a to jak vzestupně tak sestupně, čímž bude uživateli poskytnuta vysoká svoboda vyhledávání ve velkém množství sítí.
Pokud ani seřazení nebude dostatečně účinné k dosažení cílů uživatele, měl by ten být také schopen vyfiltrovat síť podle jejího identifikátoru nebo rozsahu identifikátorů.

Poslední z navrhovaných funkcí pro tento režim je schopnost listovat mezi časově sousedícími data sety (tzn. změnit výběr na nejbližší data set s vyšším či nižším časovým razítkem).

Nakonec, pro účely testování, bude prototyp aplikace obsahovat panel pro náhodné generování testovacích dat, která budou na základě stanovených parametrů vytvořena a uložena do databáze. Tato součást bude samozřejmě určena k odstranění v dalších verzích aplikace, kdy již budou dostupné sběrače dat z reálných zařízení.

9.3.2 Vlastnosti detailu

V módu detailního zobrazení informací sítě bude aplikace zobrazovat jednu heatmapu ve větší velikosti s popisky řádků a sloupců (IP adresy k ním náležící). Tato heatmapa bude umožňovat vykreslení barevné škály jak v globálním (vůči nejvyšší hodnotě ze všech sítí, stejně jako v přehledu) tak v lokálním měřítku. Použitá barevná škála bude na obrazovce pro přehlednost vykreslena (divergentní škála formátu "-Upload, 0, Download") a v ní budou zobrazeny maximální hodnoty současné sítě (max upload, max download).
Při pohybu myší po mapě by tato měla zvýrazňovat k políčku náležící řádek a sloupec pro ulehčení vyhledávání uživatelů ve vizualizaci a také by tito měli být schopni zjistit hodnotu toků v aktuálně vybrané buňce mapy.

Seznam v levé části obrazovky zde zůstává, nicméně bude obsahovat pouze položku aktuálně vybrané sítě s rozevřeným listem příslušných IP adres. Na

tomto listu bude pomocí bar chartu vizualizován podíl zátěže každé z adres na celkové (vnitřní) zátěži sítě. Pokud uživatel myší klikne na nějaký z popisků mapy, bude tomuto popisku příslušný řádek či sloupec trvale zvýrazněn, dokud na daný popisek uživatel znovu neklikne. Takto bude možné v mapě označit libovolné množství řádků a sloupců.

Nakonec bude ve stránce také vyobrazen dříve diskutovaná bar chart vizualizující agregaci zátěže ve veřejných IP adresách k síti náležících. Tento graf bude také vykreslen v aktuálně vybraném měřítku (lokální/globální).

9.4 Ovládací prvky

Implementovaná aplikace by měla obsahovat přinejmenším následujícími prvky uživatelského rozhraní:

- Tlačítko pro otevření dialogu načtení dat
- Tlačítko pro otevření dialogu generování dat
- Možnost zobrazení okna s aplikačními nastaveními uživatele
- Zobrazení okna pro osobní nastaveními uživatele (email, heslo atd)
- Možnost přechodu do detailního zobrazení přes přehled i seznam
- Tlačítka pro seřazení seznamu a přehledu sítí
- Textové pole pro filtrování rozsahu sítí

9.5 Odkazy na stav

Pro zajištění pohodlné obsluhy uživatelem a jednoduchého sdílení specifických dat by aplikace měla umět vyobrazovat informace o jejím aktuálním stavu (dataset, id detailu) a možnost okamžitého načtení tohoto stavu.
Této funkce bude dosaženo za pomocí dodatečných parametrů URL adresy, které budou indikovat aktuálně načtený data set a pokud bude aplikace v režimu detailu tak také identifikátor zobrazené sítě. Takovýto link bude možné sdílet (či opětovně načíst) pro okamžité načtení těchto údajů.

9.6 Shrnutí návrhu

V této kapitole jsme si určili jakého typu a architektury se bude naše implementace snažit držet a docílit a specifikovali funkce, které budeme od webové části aplikace požadovat a jakým způsobem bychom si je přáli ovládat.

Kapitola 10

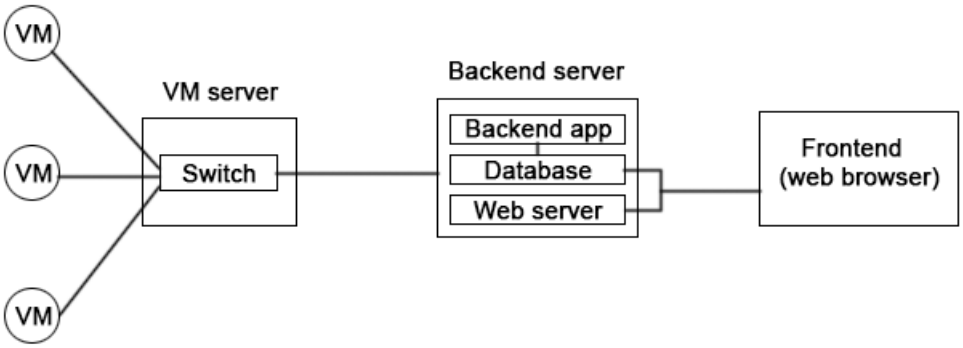
Návrh sběrače dat

V této sekci se blíže podíváme na způsoby, jakými lze získat data k zobrazení a nástroje, jež lze k této činnosti využít. Za jejich pomoci vytvoříme návrh funkčního procesu ke sběru, zpracování a přepravě dat. Přestože tento návrh nebude jako součást práce implementován, lze ho využít jako výchozí exemplář pro implementace ke sběru dat v cloudech podobné funkcionality a také jako návrh k implementaci skutečného sběrače v námi specifickém cloudu.

10.1 Popis funkcionality

Ukažme si zde znovu obrázek z kapitoly 5 zobrazující architekturu vizualizační aplikace.

Popišme si zde detailněji jakým způsobem funguje prostředí virtualizačního cloudu BigCloud.
V levé části obrázku vidíme nákres našeho případu hypervizoru (server hostující virtuální stroje). Tyto stroje využívají virtualizačního kernelu *qemu* obecně používaného v nástroji KVM [33] (ačkoli implementace bigcloudu se od tohoto nástroje abstrahuje a pracuje přímo s kernelem) pro tvorbu a zprávu virtuálních strojů.
Servery fungují na principu master-slave s využitím principů vynucené konfigurace za pomocí nástroje SaltMaster [34]. Tento program využívá kontrolního serveru (na němž nejsou vytvářené žádné virtuální stroje) ke kontrole toho, zda na všech hypervizorech v cloudu jsou vytvořené a spuštěné všechny virtuální stroje, které tam být mají (a v příslušné aktuální konfiguraci).
Databáze bigcloudu (MongoDB) obsahuje informace o všech hypervizorech a VM na nich existujících, jejich stavu (zapnutý, vypnutý), počtu procesorů, síťové konfiguraci, přidělené paměti atd. Při každé změně zasahující daný hypervizor SaltMaster provede množství v Pythonu napsaných skriptů, které porovnají aktuální stav databáze s aktuálním stavem hypervizoru a provede takové změny, které jsou nezbytné v vynucení tohoto stavu na hypervizoru (přidání, smazání, překonfigurování VM).



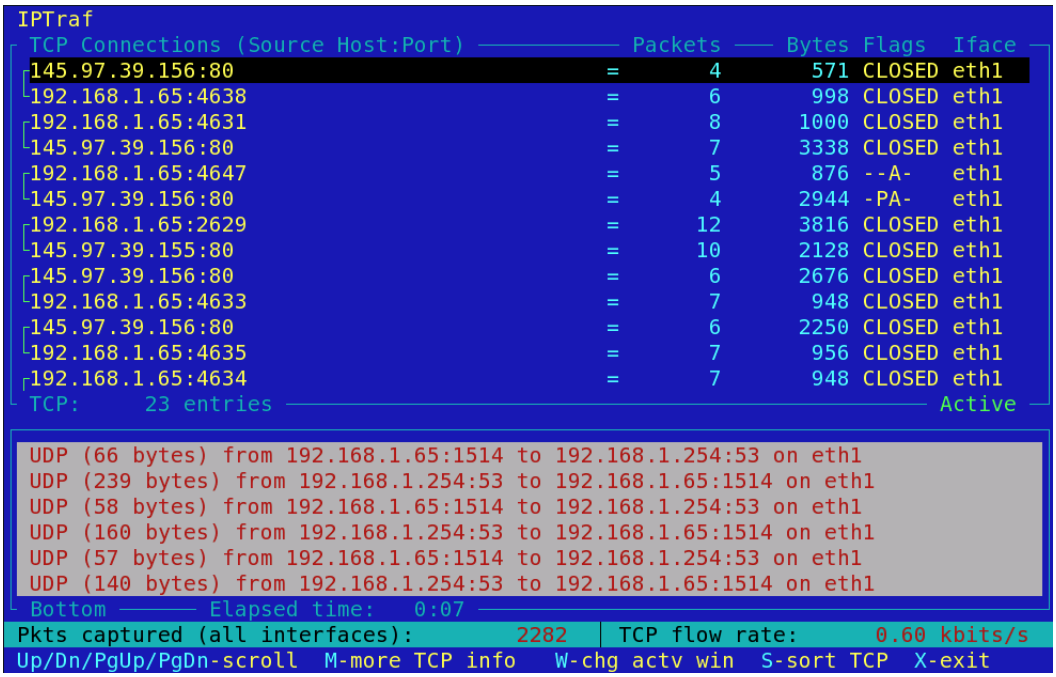
Obrázek 10.1: Obecná architektura webové vizualizační aplikace

VM tímto způsobem vytvořené k síťové komunikaci používají speciální softwarový switch, program zvaný Open vSwitch [35]. Jedná se o virtuální switch zařízení provozované na každém z hypervizorů, které se virtuálním strojům jeví a chová jako skutečný fyzický switch. K tomuto se stroje připojují pro přístup k síti.
Open VSwitch je moderní a populární volbou pro virtualizaci síťových zařízení. Jeho značná funkcionalita obsahuje mimo jiné podporu protokolu OpenFlow. Jedné se o technologii, poskytující uživateli kompletní kontrolu nad tím co se uvnitř switche děje. Po základní instalaci je chování všech portů pro všechny stroje nastaveno na "normal", čímž je emulováno chování normálního fyzického zařízení. Administrátor však může do zařízení vkládat OpenFlow pravidla, které stanovují jak bude s určitými pakety naloženo. Obecně je nutné definovat port switche pro které bude pravidlo použito (nebo skupinu či celek) a poté syntaxe nabízí téměř neomezené možnosti. Pakety je možné zkoumat na dodržení či porušení určitých vlastností (IP adresa zdroje/cíle, MAC adresy, VLAN) či flagů a na základě toho aplikovat akci jak s paketem naložit (např. přesměrovat na určitý port, zahodit, emulovat "normální"chování).
Jedná se tedy o komplexní nástroj dovolující velmi rozmanité konfigurace. Jednotlivá pravidla a počty paketů, které jim vyhověli lze v systému sledovat a na základě těchto by mohlo být možné získávat požadované informace o tocích. Tento přístup by však vyžadoval velmi drobnou konfiguraci specifických

pravidel pro každý virtuální stroj a počty takto zpracovaných paketů jsou relevantní pouze k současné relaci, nastával by tedy risk ztráty dat.
Podstatně lepším řešením pro tento účel je využitím další z funkcí nástroje a to vestavěné schopnosti monitoringu s využitím některého z existujících síťových protokolů. O této schopnosti si více povíme v další části kapitoly.

10.2 Sběr dat

S již nastíněnými principy funkcionality virtualizační a síťové části cloudu se v této části můžeme podívat na jednotlivé způsoby, které by ke sběru dat o tocích v síti bylo možné použít.



Obrázek 10.2: Terminálový výstup aplikace iptraf [36]

- tcpdump / iptraf
  Naivní řešení za použití základního linuxového nástroje tcpdump poskytující informace o paketech protékajících přes zvolené rozhraní (v našem případě virtuální switch). Použití spočívá ve zvolení rozhraní a parametrů k monitorování a program poté do terminálu po řádcích vypisuje požadované informace o paketech, které přes rozhraní prochází. Na obrazovce terminálu nám samozřejmě informace nejsou příliš k užitku, naštěstí však tento výstup lze přesměrovat do souboru ve formě logu. S tímto však nastávají další potíže.

Takto získaný soubor je k dispozici na hypervizoru samotném, nicméně je stále otevřený a aktualizovaný, nelze jej tedy dobře zpracovat ani přeposlat na aplikační server. Teoreticky by mělo být možné použít rotace souboru, jako je tomu dělané u běžného linuxového logu, tímto však zavádíme do problematiky další komplexitu. Takto získané logy by také nabývaly velké souborové velikosti, vzhledem k vysokému počtu řádků (1 paket = 1 řádek) v nich obsažených.

Vhodnějším způsobem by byla možnost přenesení výstupu programu přes síť přímo do backendové aplikace ke zpracování. Za tímto účelem bylo otestováno řešení s použitím linuxového nástroje *netcat* poskytující rozmanité TCP/IP funkcionality, včetně přenosu dat mezi servery. Klasický souborový formát k ukládání informací také nebyl příliš vhodný, jelikož je tímto způsobem nemožné zjistit, které informace už byly zpracovány a které jsou nové přidané. Toto bylo vyřešeno využitím speciálního linuxového souboru formátu *named pipe*, který se chová podobně jako jeho terminálová varianta pro přenos dat mezi nástroji, lze k němu však přistupovat jako ke klasickému souboru. Data vložená do takovéhoto souboru mají jednu velmi vhodnou vlasnost - jakmile řádek přečteme, je ze souboru odstraněn a ten tedy vždy obsahuje pouze aktuální nezpracované informace.

Testovaný proces fungoval následujícím způsobem:
Uživatel vytvořil v systému soubor formátu named pipe. Poté byl spuštěn monitorovací nástroj s přesměrováním výstupu do tohoto souboru. Jako další se na stejném stroji spustil linuxový příkaz *cat* pro čtení souboru a jeho výstup se přesměroval do nástroje netcat se zadanou adresou backendového serveru. Tento řetězec příkazů zůstal nekonečně otevřený a tedy libovolně dlouho fungující (dokud ho uživatel nebo systém neukončil), jelikož soubor formátu named pipe neobsahuje flag EOF (End Of File), který cat používá k detekci konci souboru a ukončení čtení.
Pro přijímání informací na straně backendu byl opět spuštěn nástroj netcat, tentokrát v módu odposlechu a také s parametrem spuštění aplikace, do které byly odposlechnutá data přeposlány (naše backendová dummy aplikace vypisující přijaté řádky do terminálu).
Přes relativně velké množství potřebných nástrojů je tento typ postupu fungující a data se v aplikaci objevovala, během testování však byly nalezeny poměrně závažné nedostatky. Tím menším z problémů byl fakt, že se systém čas od času rozhodl některé z nástrojů ukončit a tím celý řetěz přerušit. Bylo by tedy nutné připravit monitorovací skripty, které by takovouto událost dokázali detekovat a řetězec obnovit.
Závažnějším problémem však je neschopnost nástroje netcat přijímat více než jedno spojení zároveň, omezující efektivitu našeho řešení pouze na formát komunikace 1:1. S požadavkem na libovolné množství hyper-

vizorů v cloudu toto omezení není přípustné a jeho obejití by vyžadovalo další rozšíření procesu.

Poslední, nejkomplexnější ale zároveň velmi robustní variantou k přenosu dat v tomto řešení je namísto nástroje netcat vytvoření vlastní odesílací a přijímací C++ aplikace za pomocí rozhraní TCP/IP. Tento způsob by však vyžadoval zajištění možnosti komunikace s libovolným počtem kanálů a tedy vytváření samostatných podprocesů pro každou jednu komunikaci. Zajištění dostatečné stabilnosti a bezchybnosti obou stran aplikace by však v tomto případě přinášelo značné úsilí.

Jak je zmíněno v nadpisu této varianty, alternativně k programu tcpdump lze pro sběr dat použít nástroj iptraf, který v terminálové verzi nabízí agregované informace o tocích mezi jednotlivými IP adresami. Znovu však vystává problém přenosu této informace do aplikace, jelikož při zvolení přesměrování výstupu do souboru se dostáváme na úroveň výstupu programu tcpdump s jednotlivými řádky obsahující informace o paketech.

- Open vSwitch pravidla
  Jak již bylo zmíněno v popisu tohoto nástroje, jednotlivá definovaná OpenFlow pravidla obsahují při svém výpisu čítače počtu bytů a paketů, které tomuto pravidlu vyhověli. Bylo by tedy možné nadefinovat velké množství pravidel (1 pravidlo pro každou dvojici VM ve společné síti) pomocí kterých bychom mohli tyto aktivity sledovat.
  Navrhovaný proces by spočíval ve vytvoření skriptů integrovaných do monitorovací struktury cloudu, které by se spouštěli v pravidelných intervalech, příkazem získali výpis pravidel a za použití informací o předchozích stavech čítačů tyto informace rozparsovali a vypočteli diferenciál.
  Ačkoli se jedná o poměrně přímočarou techniku, spočívá v ní také řada nevýhod. Hlavní z těchto je zejména obrovské množství pravidel, které bychom museli pro každý VM definovat, jejichž počet by s rostoucím počtem VM rostl exponenciálně. Tato vlastnost spolu s možnou nutností úprav stávajících OpenFlow pravidel a nutnosti započtení bezpečnostních pravidel zamezující komunikace s neautorizovanými adresami je dostatečně závažná aby řešení učinila nevhodnou volbou.
  Další z nevhodných vlastností je již zmíněná relativnost hodnot čítačů vůči danému sezení a nedostupnost informací o podílech protokolů a portů na takto zaznamenané komunikaci (pokud bychom nechtěl ještě mnohonásobně zvýšit počet pravidel jejich další specifikací).

- NetFlow [37] / IPFIX [38] / sFlow protokoly [39]
  Poslední z možných řešení sběru dat je využití vestavěné monitorovací funkcionality programu Open vSwitch, kterou je možné nakonfigurovat

k využití jednoho ze tří síťových protokolů navržených pro monitorování dat.

Netflow - protokol vyvinutý firmou Cisco v roce 1996 za účelem monitorování datových toků v síti na zařízeních této firmy. Jedná se o patentovanou technologii a její hlavní nevýhodou je zaměření na zařízení Cisco.

IPFIX - technologie jejíž zkratka značí "IP Flow Information eXport"byla vyvinuta na základě protokolu NetFlow v9 a na její tvorbě se podíleli někteří lidé, kteří tvořili samotný NetFlow protokol. Jedná se o modernější řešení jehož cílem je rozšíření do co nejvyššího možného počtu zařízení pro zajištění vysoké kompatibility.
Zatímco devátá verze protokolu NetFlow poskytuje 79 typů polí ke sběru informací o tocích, IPFIX nabízí těchto 79 k zajištění perfektní zpětné kompatibility a toto rozšiřuje na celkových 238 datových polí. Jedná se tedy o moderní výkonnější volbu se zpětnou kompatibilitou.
Jak IPFIX tak jeho předchůdce NetFlow fungují na principu agregace dat.

sFlow - jak význam této zkratky "sampled flow"napovídá, protokol se na rozdíl od předchozích dvou příkladů zabývá poskytnutím informací o tocích za využití vzorkování. V daných časových intervalech jsou tedy sbírání vzorky aktivity paketů na zařízení a za jejich využití je vytvořen předpoklad skutečného chování dat. Přesnost takto získaných informací závisí na délce intervalů mezi sbíráním vzorků a platí že čím častěji toto děláme, tím pravděpodobněji budou informace přesné. Za častější vzorkování však platíme vyššími požadavky na výkon.
Protokol byl navržen s cílem vysokého výkonu za cenu možné nepřesnosti informací.

Nejvhodnější volbou z těchto tří nabízených je pro naše účely protokol IPFIX, který nabízí vysoké možnosti sběru informací, kompatibility a přesnosti výstupů.
Co se odesílání informací týče, o toto se po nakonfigurování stará Open vSwitch sám, není tedy nutné pro tento úkon vynakládat žádné další úsilí. Je však třeba nějakým způsobem tyto data zachytit a zpracovat.
Na internetu můžeme najít množství kolektorů pro data poskytnuté těmito protokoly. Jako atraktivní volba se jeví nástroje *nfcapd* [40] sloužící jako backendový kolektor pro webovou vizualizační aplikaci síťové aktivity NfSen, kterou jsme si představili v kapitole 3. Popis tohoto nástroje říká následující

> "Collects the netflow data, sent from exporters and stores the flow records into files. Automatically rotates files every n minutes. (typically every 5 min) The netflow versions mentioned above are read transparently Multiple netflow streams can be collected by a single or collector. nfcapd can listen on IPv6 or IPv4. Furthermore multicast is supported."

spolu s tvrzením o kompatibilitě s protokolem IPFIX se tak jedná o ideální řešení.

Po zvážení všech vypsaných možností se nakonec jako nejlepší přístup k problému jeví využití již stávajících nástrojů a protokolů a tedy schopnosti programu Open vSwitch poskytovat monitoring své aktivity a to za pomocí protokolu IPFIX a sběrače dat nfcapd.
Tímto máme zajištěný sběr dat i jejich transport na cílový server s vizualizační aplikací.

10.3 Zpracování dat

V tomto okamžiku je třeba data nacházející se na zdrojovém serveru ve formátu množství souborů nějakým způsobem zpracovat a předat backendové části vizualizační aplikace. K tomuto účelu je vhodné využít skriptu napsaného v jazyce Python, který bude v pevně stanovených intervalech systémem spuštěn, zpracuje aktuálně připravené informace o tocích, převede je do požadovaného formátu (viz formát souboru JSON v kapitole 6) a zašle tento soubor na poskytnuté API naší aplikace.

Ke zpracování dat získaných výše popsaným způsobem, která jsou uložena v binárním formátu využijeme programu *nfdump* [40], který je součástí backendových nástrojů vizualizační aplikace NfSen. Příslušná dokumentace nástroj popisuje takto:

> *"Nfdump reads the netflow data from one or many files stored by nfcapd. It's filter syntax is similar to tcpdump (pcap like) but adapted for netflow. If you like tcpdump you will like nfdump. nfdump displays netflow data and/or creates top N statistics of flows, bytes, packets. nfdump has a powerful and flexible flow aggregation including bi-directional flows. The output format is user selectable and also includes a simple csv format for post processing."*

Skript tedy tuto aplikaci vyvolá, výstup přečte a zpracuje do souboru JSON a opatří ho časovým razítkem. Výsledek poté zašle do vizualizační aplikace.

10.4 Shrnutí

V této kapitole jsme si představili principy, na základě kterých pracuje správa virtuálních strojů našeho cloudu a virtuální switch, který každý z hypervizorů používá.
V další části byly zanalyzovány možnosti sběru dat a ustanovena vhodnost použití implicitní monitorovací funkce programu Open vSwitch se síťovým protokolem IPFIX k tomuto účelu určenému. Takto nasbíraná data budou zachycena aplikací nfcapd a zpracována Python skriptem využívajícího parsovacího nástroje nfdump.
Výstupem skriptu bude JSON soubor, který bude zaslán na API naší aplikace a tou uložen do databáze.

Kapitola 11

Implementace backendu

Velké množství informací pokrývající funkci a schopnosti backendové části bylo již popsáno v návrhu, v této části si některé tyto informace ještě upřesníme a podíváme se na bližší technologické detaily samotné implementace, počínaje od instalace a nastavení serveru až po specifické části frameworku Symfony.

11.1 Použitý server a jeho charakteristiky

Jako zázemí pro celou tuto část bylo zvoleno použití jednoho z virtuálních strojů na samotném cloudu kterým se tato aplikace zabývá. Lokace aplikace uvnitř cloudové sítě zjednoduší budoucí implementace sběru dat a ušetří nás nutnosti transferu velkých množství dat po internetu. Díky přítomnosti nástroje na cloudové síti se také zvyšuje bezpečnost a integrita celého systému. V případě potřeby by aplikace však byla schopna fungovat z jakékoli jiné serverové lokace.

Ve vývojovém prostředí cloudu byl tedy vytvořen jeden nový virtuální stroj s následujícími specifikacemi:
Počet virtuálních procesorů: 1
Velikost paměti RAM: 1GB
Počet disků: 1
Velikost disku: 30GB
Síťové rozhraní s 1 veřejnou IP adresou (185.88.72.9)

Na tento VM byla projedena čistá instalace Linuxového systému Debian 8 Jessie v serverovém provedení (bez grafického rozhraní) obsahující pouze SSH server pro možnost vzdáleného připojení ke stroji.

Na tomto systému byly následně provedeny aktualizace a nainstalovány základní systémové nástroje včetně webového serveru Apache2 ve verzi 2.4.10 (Debian).

Dále byly nainstalovány databáze MySQL (verze 14.14), PHP 5.6.(včetně php-dev nástrojů nutných k instalaci rozšíření pro ext-mongo) a MongoDB verze 3.6.4.

Pro umožnění PHP pracovat s MongoDB nástroji bylo nezbytné na tento stroj nainstalovat příslušné php rozšíření ext-mongo pomocí konzolového příkazu

```
pecl install mongo
```

a do příslušných konfiguračních souborů php v "/etc/php5/cli/php.ini"a "/etc/php5/apache2/php.ini"přidán řádek aktivující toto rozšíření

```
extension=mongo.so
```

V těchto souborech je také nutné správně nakonfigurovat informaci o časové zóně pro správnou funkci frameworku Symfony a tedy upravit příslušný řádek na (či jiné vhodné nastavení)

```
date.timezone = Europe/Prague
```

Po provedení těchto změn je nutné restartovat službu apache2.

11.2 Konfigurace webového serveru

Po nainstalování nezbytných nástrojů byla provedena konfigurace webového serveru a přenesení souborů aplikace do webové složky ("/var/www/netvis"). V konfigurační složce programu Apache2 ("/etc/apache2") byla v podsložce "sites-available"vytvořena konfigurace virtuálního hosta pro naši aplikaci, která vypadá následovně

```
<VirtualHost *:80>

        ServerName 185.88.72.9/netvis
        ServerAdmin admin@localhost
        DocumentRoot /var/www/netvis/web
            <IfModule mod_rewrite.c>
    Options +FollowSymlinks
    RewriteEngine On

    # Explicitly disable rewriting for front controllers
    RewriteRule ^app_dev.php - [L]
    RewriteRule ^app.php - [L]

    RewriteCond %{REQUEST_FILENAME} !-f

    # Change below before deploying to production
    #RewriteRule ^(.*)$ /app.php [QSA,L]
```

```
    RewriteRule ^(.*)$ /app_dev.php [QSA,L]
```

</IfModule>

Tento soubor instruuje webový server aby veškerá příchozí webové spojení směroval do složky naší aplikace, odkud Symfony práci převezme.
Součástí nastavení je také dynamické přepsání url adresy, která ve výchozím stavu obsahuje název složky s webovým obsahem a aplikační Symfony PHP soubor (app.php pro produkční verzi, app_dev.php pro vývojovou) a odkaz na aplikaci by tedy vypadal zhruba takto:

185.88.72.9/web/app.php/

S použitím konfigurace výše dostáváme "hezčí"tvar této adresy ve tvaru:

185.88.72.9/

Po zkopírování souborů symfony do této specifikované složky ještě bylo třeba vyčistit cache frameworku. To lze udělat přepnutím se v terminálu do složky aplikace ("/var/www/netvis") a provedením následujícího příkazu

```
php bin/console cache:clear -e=dev
```

část "dev"nám říká že se jedná o vyčistění cache pro vývojovou verzi aplikace, učinění stejné akce pro verzi produkční lze udělat nahrazením slova "dev"v příkazu slovem "prod".

Jelikož Symfony vyžaduje práva do této složky (specificky podsložky var) zapisovat toto bylo provedeno změnou přístupových práv složky "var"příkazem.

```
chmod -R 777 var
```

Pravděpodobně by existovalo lepší a bezpečnější řešení tohoto problému, pro potřeby aplikace bylo však toto řešení dostatečné. Je ovšem třeba mít na paměti, že tuto změnu práv je nutné provést pokaždé, když z konzole čistíme cache Symfony (jelikož je složka "var"odstraněna a nahrazena novou s výchozími právy).

11.3 Konfigurace databází

Databáze MySQL byla po své instalaci zabezpečena skriptem spuštěním příkazu "mysql_secure_installation"který odstranil nebezpečné testovací vlastnosti.
Za pomocí konzolového nástroje mysql byla vytvořena nová databáze jménem "networkVisBackend"a nový databázový uživatel "symfony". Tomuto uživateli byla dána plná práva nad touto databází a přístup z lokálního stroje

"localhost". (Databázi samotnou by nebylo nutné vytvářet a Symfony by to dokázalo udělat za nás, byla však žádané vytvořit nového uživatele specifického pro aplikaci a tomuto přidělit práva pouze do této databáze, proto byl proveden přístup popsaný výše).

Co se druhé databáze, MongoDB, týče, při instalaci operačního systému byl souborový systém připraven takovým způsobem, aby operační systém měl k dispozici 10GB z celkových 30 a zbylé místo bylo přidělené diskovému oddílu připojeného do složky "/data". Tato složka je tedy určená k ukládání velkých množství dat a je provedena tímto způsobem, aby bylo jednoduché v případě potřeby navýšit paměť datového oddílu disku bez ovlivnění zbytku systému. V konfiguračním souboru databáze ("/etc/mongod.conf") byla tedy provedena úprava složky pro uchování dat na "/data/mongodb", tato podložka v "/data"vytvořena a její uživatel a uživatelská skupina změněny na mongodb. Poté byla databáze restartována příkazem

```
service mongod restart
```

V MongoDB byla také vytvořena databáze netvis pro použití v aplikaci.

11.4 Konfigurace Symfony

V této chvíli máme složku aplikace vytvořené ve frameworku zkopírovanou na serveru a připravenou k použití. Nejprve je však nutné získat nezbytné přídavky a vyplnit některé konfigurační soubory.

11.4.1 Použití konzole

Již v předchozí části této kapitoly jsme si ukázali použití Symfony konzole k vyčistění cache frameworku. Tento nástroj nám však umožňuje mnohem více a poskytuje nápomocné funkce k aktuálně nainstalovaným balíkům.

Mezi některé z užitečných příkazů patří například tyto (příklady předpokládají že se uživatel v terminálu nachází ve složce frameworku, jinak je nutné patřičně upravit cestu k souboru konzole)

```
// Vypise seznam dostupnych prikazu
php bin/console list

// Spusti lokalni server na zadanem portu (pro lokalni vyvoj)
php bin/console server:start <port>

// Prikazy pro vycisteni a zahrati cache (pred deployem)
php bin/console cache:clear
php bin/console cache:warmup
```

```
//Nastroje doctrine k prace s databazi
//vytvori / smaze MySQL databazi
php bin/console doctrine:database:create
php bin/console doctrine:database:drop --force

//Overi spravnost MySQL entit
//(bundle referuje ke jmenu bandlu s aplikaci, u nas AppBundle)
php bin/console doctrine:schema:validate

//Updatuje schema MySQL databaze podle entit
php bin/console doctrine:schema:update --force

//Vygeneruje gettery a settery pro tridy entity (MySQL)
php bin/console doctrine:generate:entities <bundle>

//Vygemeruje gettery a settery pro tridy dokumentu (MongoDB)
php bin/console doctrine:mongodb:generate:documents <bundle>
```

Toto byly pouze některé z příkladů použití konzole frameworku Symfony, nástroje takto poskytnuté jsou však velmi mocné a jejich použitím si lze ušetřit velké množství práce.

11.4.2 Nainstalované doplňky

Již v návrhu jsme diskutovali, že do frameworku vedle základních součástí bude nezbytné doplnit další komponenty pro zajištění nezbytné funkcionality. Toto provedeme za použití také předem zmíněného nástroje composer.
Byl tedy upraven soubor composer.json nacházející se ve složce aplikace a do části

```
"require": {
...
}
```

byly doplněny jména těchto 3 balíků

```
"doctrine/mongodb-odm-bundle": "^3.0",
"friendsofsymfony/rest-bundle": "^2.2",
"jms/serializer-bundle": "^2.0"
```

které poskytují funkce mongodb ovladače, REST API a serializéru.

Tyto balíky byly nainstalovány následujícím příkazem ze složky aplikace (pro tuto akci byl lokálně nainstalován program composer podle pokynů z dokumentace Symfony):

```
php composer.phar update
```

Podle použité verze Symfony může být také nutné ujistit se, že jsou tyto balíky zaregistrovány v souborech "app/autoload.php"a "app/AppKernel.php".

Konfigurace takto nainstalovaných nástrojů byla provedena doplněním souboru "app/config/config.yml"o tyto parametry:

```
parameters:
    locale: en
    jms_serializer.serialized_name_annotation_strategy.class:
          JMS\Serializer\Naming\IdenticalPropertyNamingStrategy

doctrine_mongodb:
    connections:
        default:
            server: "%mongodb_server%"
            options: {}
    default_database: netvis
    document_managers:
        default:
            auto_mapping: true

fos_rest:
   routing_loader:
       default_format: json
   view:
        view_response_listener: force
        formats:
            json: true
            xml:  true
   body_converter:
         enabled: true
```

11.4.3 Připojení databází do Symfony

Databáze jsou připravené a ovladače nainstalované a nakonfigurované, stačí již jen otevřít soubor "app/config/parameters.yml"a vyplnit do něj přístupové parametry do našich dvou databází. Obsah souboru vypadá takto (heslo MySQL bylo vynecháno z důvodu bezpečnosti):

```
parameters:
    database_host: 127.0.0.1
    database_port: null
    database_name: networkVisBackend
    database_user: symfony
    database_password:
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    secret:
    mongodb_server: "mongodb://localhost:27017"
```

11.4.4 Zabezpečení

Posledním z upravených natavení frameworku je nastavení zabezpečení (firewall aplikace vyžadující login), které využívá dříve definované databázových entit User a Role, formulářové html šablony pro login (nalezené ve složce Resources/views uvnitř aplikace) a následujícího nastavení v konfiguračním souboru "app/config/security.yml"

```
security:
    encoders:
        AppBundle\Entity\User:
            algorithm: bcrypt

    role_hierarchy:
        ROLE_ADMIN:       [ROLE_USER, ROLE_ALLOWED_TO_SWITCH]

    providers:
        administrators:
            entity: { class: AppBundle:User, property: username }

    firewalls:
        secured_area:
            pattern:    ^/
            anonymous: ~
            form_login:
                login_path: login
                check_path: login_check
            logout:
                path:   /logout
                target: /
```

```
            switch_user: true

    access_control:
        - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/, roles: [IS_AUTHENTICATED_FULLY,
                              IS_AUTHENTICATED_REMEMBERED] }
```

Tento konfigurační soubor nám definuje uživatelskou entitu k použití pro vyhledání uživatele v databázi (User) a algoritmus pro hashování hesel (bcrypt). Dále je v souboru definována hierarchie rolí (admin má automaticky také roli uživatele) a nakonec definice zabezpečených oblastí aplikace (podle url) a přístupových práv.
Ze souboru lze vidět že všechny součásti aplikace kromě loginu se v současném nastavení nacházejí za firewallem, který dovoluje přístup pouze přihlášeným uživatelům (v budoucí verzi aplikace by pravděpodobně bylo vhodné z tohoto firewallu vyloučit cestu k API a provádět autentizaci těchto dotazů jiným způsobem, například pomocí API klíče).

11.5 Shrnutí implementace backendu

V sekci jsme získali informace o programech použitých k běhu aplikace, jejich verzích a použitých konfiguracích a základní informace použitelné pro budoucí údržbu, běh a rozšíření aplikace.
Mimo výše uvedené postupy byly během implementace backendu také vytvořeny všechny třídy dokumentů, entit, controllerů a HTML šablon takovým způsobem, aby byla zaručena požadovaná funkcionalita specifikovaná v návrhu této části aplikace.

Kapitola 12

Implementace frontendu

S backendovou částí aplikace implementovanou je nyní čas podívat se blíže ne detaily části frontendové běžící v internetových prohlížečích uživatelů. Jak již bylo popsáno v návrhu, tato část skládá převážně z javascriptového kódu který za pomocí informací získávaných z backendu pomocí API provádí dynamické generování obsahu stránky podle aktuálních akcí uživatele. Naprostá většina kódu této stránky se bude nacházet v souboru "main.js"v aplikační složce "web".

12.1 Dotazy na API

Rozhraní pro dotazy máme v tuto chvíli hotové a je třeba zabývat se způsobem jakým ho z webového prohlížeče použít. Naštěstí pro tvorbu aplikace používáme javscriptový framework jQuery, který disponuje množstvím na javascriptu založených nástrojů pro ulehčení a zefektivnění práce. Jedním z těchto nástrojů je také funkce Ajax určená k provádění asynchronních HTTP požadavků.
Slovo "asynchronní"je pro tento účel podstatné, jelikož znamená že se dotaz zpracovává v pozadí zatímco vykonávání kódu pokračuje, nemusíme tedy při používání čekat na odpověď serveru zatímco stránka nebude nic vykonávat.
Pro uvedení příkladu takového dotazu takto vypadá volání pro získání specifického data setu jehož časové razítko je uložené v proměnné "timestamp".

```
$.ajax({
  type:'GET',  // Typ dotazu
  url: "http://185.88.72.9/api/data/" + timestamp, // url
  dataType: "json", // Typ dat k zaslani/prijmuti
  success: function(data){
     // .. kod provedeni po uspesnem dokonceni dotazu "data"
     }
 });
```

Takovýmto způsobem jsou z kódu provedena volání kdykoli je nezbytné z uživatelova prohlížeče komunikovat se serverem. Uživateli je ve většině případů (pokud je to vhodné či žádané) zobrazeno upozornění že provádí komunikace se serverem ve formě animované ikony, která zmizí ve chvíli dokončení požadavku.

12.2 Použití d3.js

Tato javascriptová knihovna funguje na principu vázání dat k DOM elementům HTML stránky a jejich využití k vykreslení (nejen) vizualizací.

Vykreslení heatmapy tímto způsobem je jednoduché, je jen nutné správně si pro tento účel předzpracovat. Zjednodušeně řečeno při volání "vázací"funkce do d3.js vkládáme pole s daty a pro každou jednu položku v poli bude vytvořen jeden uživatelem specifikovaný element (pokud v daném výběru neexistuje žádný takovýto element je jich vytvořeno tolik kolik je prvků v poli, v opačném případě jich je vytvořeno pouze takové množství kolik chybí do dorovnání počtu prvků v poli).
Potřebujeme tedy získat pole, které bude obsahovat jeden javascriptový objekt pro každou buňku heatmapy a na základě těchto informací necháme knihovnou vytvořit SVG plátno do kterého budou vykresleny čtverce jejichž pozice a barva je specifikována v přivázaném objektu.
Příklad informací uložených v jedné buňce heatmapy vypadá takto

```
{
  isUpload: false,
  proto: {
        tcp: 95712,
        udp: 136512,
        other: 287
  },
  sourceIP: "192.168.0.5",
  sumDown: 232511,
  sumUp: 0,
  targetIP: "192.168.0.6",
  x: 6,
  y: 5
}
```

Některé z těchto informací jsou využity pro pozdější účely (vypsání informací o buňce po najedí myši) zatímco jiné jsou nezbytné pro vykreslení heatmapy, například parametry "x"a "y"podle kterých je při zpracování vypočítávána pozice buňky v heatmapě, nebo proměnné "isUpload"a "sumDown"které určují kterou barevnou škálu použít a s jakou hodnotou.

Podobným způsobem je vázání dat a vykreslování provedeno i ve všech ostatních částech stránky které tuto knihovnu využívají (jako jsou například bar charty v listu nalevo či graf zátěže veřejných IP adres ve stránce detailu - animace tohoto grafu je také vytvořena knihovnou d3.js).

12.3 Barevná škála

Použití správné barevné stupnice a vhodné metody interpolace je v každé vizualizační aplikaci velmi podstatnou částí, která rozhoduje o schopnosti aplikace podávat kvalitní informace.
Pro účely této aplikace bylo třeba vytvořit dvě barevné škály, pro upload a download, které začínají od nuly a mají dynamicky nastavenou horní mez podle aktuálních potřeb. I zde se nástroj d3.js jeví velice užitečný se svou metodou *d3.scaleLinear*, která po nastavení vstupního (pole rostoucích čísel) a výstupního (pole barev stejné velikosti jako vstupní) rozsahu dokáže pro vložené číslo navrátit interpolovanou barevnou hodnotu dle zadané stupnice.
K zajištění maximální kvality výstupu pro účely vizualizace byla navíc tato funkce nastavena pro interpolaci v barevném prostoru HCL, který nabízí vyšší schopnost rozlišení barevných změn na stupnici navržené pro lidský zrak.

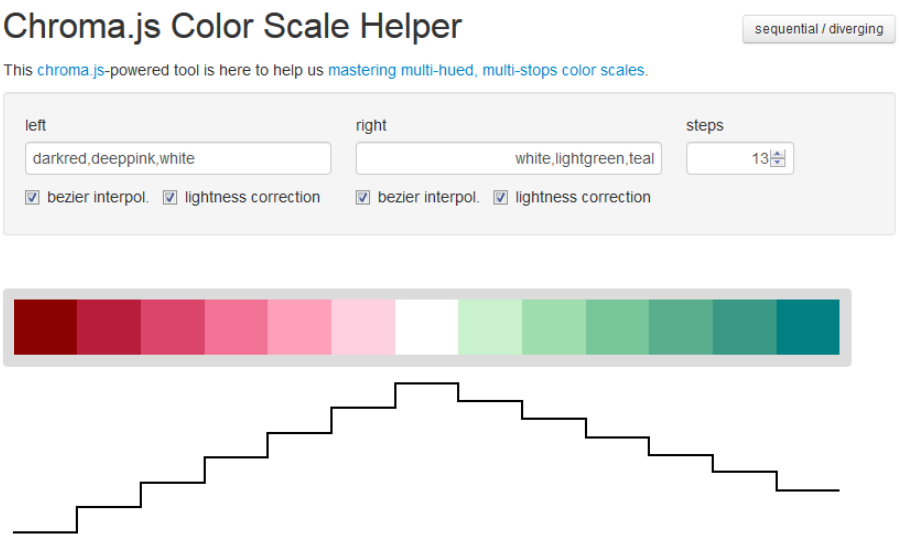
Číselné hodnoty rozsahu funkce jsou nastaveny od 0 do lokálního nebo globálního maxima dané položky (upload / download) a pro získání kvalitních barev mezi kterými interpolovat bylo použito webového nástroje Palettes [41] určenému ke generování barevných palet vysoké kvality pro použitá ve vizualizaci. Tento nástroj je založen na javascriptové knihovně určené k manipulaci barev Chroma.js a Bézierovy interpolace k získání co možná nejlepšího výsledku.

Vygenerované vstupní barvy z požadovaného rozsahu můžeme vidět na obrázku 12.1 ve formě jedné divergentní barevné škály (která byla podél bílé rozdělena na dvě).

V této divergentní formě (ačkoli obsahující interpolaci) je tato škála také vykresleno v detailním náhledu na síť, kde v ní jsou navíc vyznačeny lokální maxima dané sítě a poloha v mapě aktuálně vybrané hodnoty.

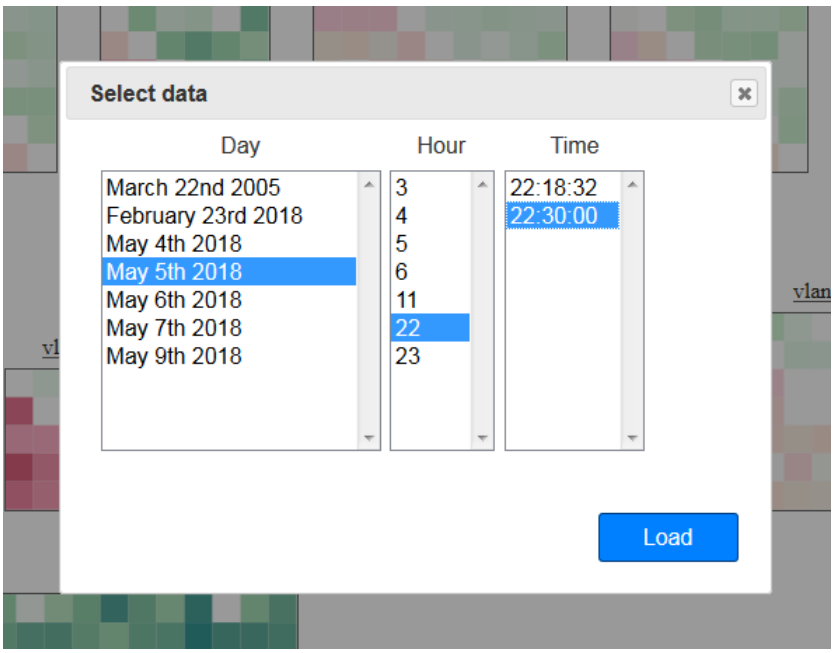
12.4 Načítání dat

Načítání dat bylo v aplikaci implementováno formou otevíracího dialogového okna (obrázek 12.2), která je aktivováno po kliknutí na příslušné tlačítko v horním panelu obrazovky.
Při otevření tohoto okna jsou z databáze načteny údaje o všech existujících časových razítkách, které frontendová část aplikace nadále rozdělí do dnů, hodin a konečně celé časové hodnoty razítka ve dni.



Obrázek 12.1: Vygenerovaná barevná škála [41]

Tyto výběrové panely dynamicky mění svou nabídku podle hodnoty vybraných v listech nalevo od nich a dochází tak k efektivní filtraci velkého množství časových dat do přehledných skupin.



Obrázek 12.2: Dialogové okno pro načítání dat

12.5 Filtrování rozsahu sítí

Poslední částí na kterou se v této kapitole podíváme je panel pro filtrování zobrazených sítí. Tento se nachází v levém horním rohu aplikace (a funguje jen v režimu přehledu, jelikož pro detail nemá význam) a skládá se z textového pole.
Uživatel může do pole vložit vzor filtrování skládající se buďto ze samotného čísla (poté jsou v seznamu a přehledu ponechány jen sítě jejichž tag začíná tímto číslem), čísla a pomlčky (v takovém případě budou ponechány všechny hodnoty od daného čísla nahoru) nebo čísla, pomlčky a čísla (například "5-30", kdy bude ponechán jen rozsah mezi těmito dvěma čísly včetně).

Veškeré změny se promítají zároveň jak do seznamu, tak do přehledu všech heatmap a provádějí se pokaždé když textové pole zaznamená změnu jednoho znaku (přidání / smazání). Jelikož se jedná o relativně náročnou operaci, zejména pro větší množství nabízených sítí, je tato funkce dynamičnosti omezena maximálním počtem 500 sítí. Při překročení tohoto limitu se vedle textového pole zobrazí vyhledávací tlačítko, kterým je třeba dotaz potvrdit k zachování dostatečného výkonu aplikace.

Tato funkce byla implementována na základě regulárních výrazů, pro které je prováděna smyčka nad všemi elementy seznamu a přehledu, výraz vyhodnocen na shodu a v případě neshody je element skrytý za pomocí jQuery.

Součástí tohoto panelu jsou také čtyři tlačítka označená ikonami T+, T- a D+, D- které slouží k seřazení položek seznamu (a přehledu) podle tagu od největší položky, tagu od nejmenší položky, datové zátěže od největší položky a datové zátěže od nejmenší položky.

12.6 Shrnutí implementace

Tato kapitola se blíže zabývala detaily implementace kódů pro provádění API dotazů z webového prohlížeče, jakým způsobem byla v implementaci použita vizualizační knihovna D3.js, způsobem kterým byla vytvořena barevná škála pro vykreslení datové zátěže a rozebráním detailů implementace některých zajímavých funkcí uživatelského rozhraní.

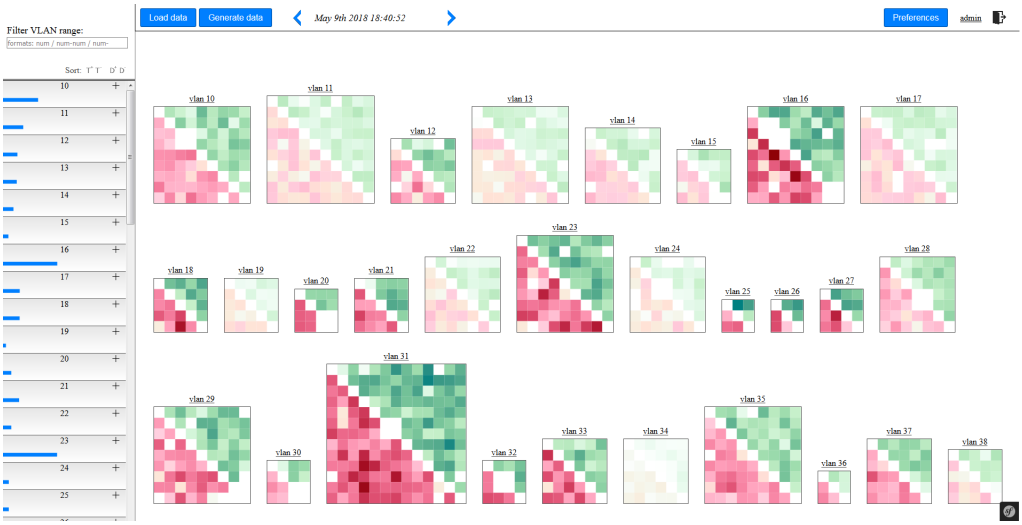
V této části jsme s implementační částí práce tedy hotový a prototyp aplikace je vytvořený a funkční. V další kapitole se podíváme na to, jaké tento prototyp podává výkony v různých prohlížečích a pokusíme se odhadnout do jaké úrovně datové zátěže si aplikace zachová schopnosti plynulého chodu.

Kapitola 13

Vytvořená aplikace

S implementací úspěšně za námi, věnujme tuto kapitolu pohledu na obrázky výsledků a jejich stručnému popisu.

13.1 Přehled



Obrázek 13.1: Obrazovka přehledu v prototypu aplikace

Úvodní obrazovkou této webové stránky (za předpokladu že má uživatel aktivní preferenci načtení nejnovějších dat při spuštění) je přehled heatmap všech sítí z aktuálně vybrané datové sady.
Tyto heatmapy zabírají většinu prostoru stránku, jelikož jsou středem zájmu uživatele. V levé části obrazovky poté vidíme stejná data uspořádaná v seznamu s vizualizací celkové zátěže. Kliknutím na položku tohoto seznamu jí lze rozevřít k zobrazení výpisu IP adres v sítí obsažených a jejich podílu na

celkové zátěži sítě (zobrazené jako modrý bar v položce, relativní vůči síti s maximálním zatížením).
Kliknutím na symbol "+"v položce lze také přejít do režimu detailu dané položky (stejný efekt jako kliknutí na její heatmapu).

Nad seznamem se dále nachází textové pole pro rozsahové vyhledávání v položkách, které ovlivňuje jak seznam, tak zobrazené heatmapy. Pole přijímá výrazy ve formátu čísla pro výpis všech sítí jejichž identifikátor tímto číslem začíná nebo dvou čísel oddělených pomlčkou pro výpis všech sítí v tomto rozsahu. Také je možné použít pouze výraz čísla zakončeného pomlčkou, který vypíše všechny sítě od daného čísla výše.

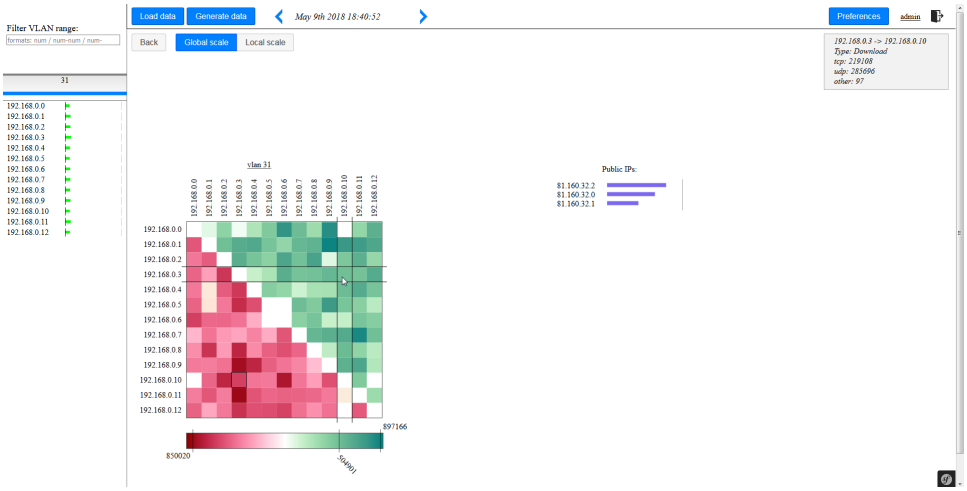
Ve vrchní části stránky se poté nachází ovládací panel, umožňující přístup k dialogu načítání dat (viz. obrázek 12.2), generátoru dummy dat a nastavení uživatele a jeho preferencí (preference po stisknutí příslušného tlačítka, osobní nastavení po kliknutí na jméno uživatele).

Vrchní část obrazovky slouží také k zobrazení informace o aktuálně vykreslené datové sadě a nabízí šipky rychlé změny na data sety tomuto nejbližší. V neposlední řadě se pak v tomto panelu nachází také tlačítko odhlášení z aplikace.

13.2 Detail

Po přepnutí aplikace do režimu detailu můžeme vidět heatmapu stejnou jako v přehledu, ačkoli ve větším měřítku a s vyobrazenými popisky řádků a sloupců. Tato vizualizace má v tomto módu rozšířenou funkcionalitu o schopnost interakce s kurzorem myši, kterým může uživatel po mapě přejíždět a získávat tak zvýraznění aktuálního řádku, sloupce, buňky zrcadlené podle diagonály a informace o aktuální hodnotě buňky.
Informace o políčku v aktuální lokaci kurzoru jsou zobrazeny na dvou místech této stránky, jednak v pravém horním rohu, kde jsou vypsány detaily o podílu protokolů na celkové zátěži, stejně jako typ toku (upload/download) a jeho účastníky (IP adresy).
Současně se také aktualizuje informace v pod mapou vykreslené barevné škále, kde je vykreslena pozice celkové zátěže políčka na barevné (spolu s údajem o této agregované hodnotě).

Pokud uživatel klikne na některý z popisků adres vedle mapy, může tímto daný řádek či sloupec také zvýraznit trvale (dokud na tento popisek neklikne znovu).
V obrazovce lze navíc vidět také tlačítka pro volbu globálního či lokálního barevného rozsahu (po jejíž kliknutí se obrazovka překreslí) a tlačítko pro návrat



Obrázek 13.2: Obrazovka detailu v prototypu aplikace

zpět do přehledu.
Na pravé straně heatmapy se také nachází seznam veřejných IP adres této sítě a vizualizace jejich agregované zátěže (po přejetí myší po této položce se v pravém horním rohu zobrazí detaily).

Levý seznam se v tomto režimu stále nachází, jediná zobrazená položka zde však pouze síť samotná, která je navíc výchoze otevřená pro zobrazení maximálního počtu informací.

Kapitola 14

Výkonnostní testování výsledné aplikace

Prototyp naší aplikace je v tuto chvíli již funkční a plně vizualizující, pro běžné používání je však také nutné, aby aplikace byla schopna zpracovávat požadavky uživatele v rozumných časech.
Vzhledem k webové architektuře nástroje je při použití nutné počítat s potřebou čekat na dokončení přenosu souboru obsahující požadovaná data, který se nachází na vzdáleném zařízení. Tuto činnost však aplikace provádí pouze ve chvíli kdy došlo ke změně používaných dat, což je akce jejíž frekvence se při běžném používání nepředpokládá nijak vysoká, krátká časová prodleva před dokončením akce je tedy přijatelná (a neodstranitelná).
V této části se podíváme na to, jakým způsobem aplikace zvládá přenos a zpracování dat při změně data setů a zda-li je plynulost používání omezena nějakým vrchním limitem počtu sítí, které je ještě možno zobrazit.

14.1 Známé technologické limitace současné verze implementace

Před započetím jakýchkoli testů je třeba zmínit, že námi používaná databáze pro ukládání vizualizačních informací, MongoDB, trpí architektonickou limitací maximální velikosti ukládaného souboru do 16 MB. Tento limit je přítomen z důvodu zachování optimálního výkonu a paměťové efektivity, omezuje nám však v základní implementaci množství informací které je možné uložit pro jeden data set.
Databáze samotná podporuje způsoby jakými do ní uložit soubory velikosti větší než tato, je však nutné využít speciální nástroje a přístupy k rozdělení souboru a uložení jeho jednotlivých součástí samostatně. Vzhledem k vyšší komplexnosti takovéto implementace, navrhované komprese ukládaného souboru (viz. kapitola o prototypu) a dostatečné velikosti limitu pro data střední

až velké velikosti byla aplikace vytvořena základním způsobem bez dělení souborů.
Během implementace a testování bylo zjištěno, že současná verze aplikace využívající tento způsob uložení je schopna spolehlivě pojmout data obsahující až 600 virtuálních sítí o velikosti 2 až 8 adres s pokrytím vnitřních spojů mezi adresami (existencí záznamu pro tok mezi dvěma adresami) nastaveným na 95%. Tento počet se ještě podstatně zvyšuje pokud omezíme procentuální šanci existence toků nebo a/nebo počet adres v sítí obsažených.

Vzhledem k tomu, že poslední autorovi známá velikost virtuálních sítí rezervovaných pro využití v cíleném cloudu se pohybovala okolo 500 (s tím že skutečně vytvořených a tedy data generujících sítí bylo podstatně méně), byla tato kapacita uznána jako dostatečně velká pro zamýšlené použití.

14.2 Průběh testování

Jak již bylo počátkem této kapitoly řečeno, zaměříme se s našimi testy na měření časů, které aplikace potřebuje k získání dat ze serveru a doby, za kterou je tyto data schopna zpracovat a vykreslit na obrazovku, samozřejmě společně s měřením času potřebným pro vykonání celé operace přepnutí mezi data sety (součet těchto dvou hodnot - získání dat a jejich zpracování).

Do příslušných částí kódu byly vloženy časovače, které jsou při změně zobrazovaných dat (skrze dialog "Load data") aktivovány a po dokončení operací vypisují do konzole prohlížeče časy, které tyto operace zabraly (v milisekundách).
Přes započetím testování bylo pro tento účel náhodně vygenerováno množství data setů v různých velikostech (30, 100, 200, 300, 400, 500 a 600 sítí) a ve dvou různých velikostech procentuální existence spojů (50% a 95%). Tyto sety byly poté v prohlížeči načítány a získané časy uloženy.

Testování probíhalo na třech moderních webových prohlížečích:

- Firefox verze 60.0.1 (64 bitů)
- Google Chrome verze 66.0.3359.181 (64 bitů)
- Internet Explorer verze 11.0.9600.19002 (64 bitů)

Parametry počítače na kterém byla měření provedeny jsou následující:

OS: Microsoft Windows 7 Professional SP1 64bit
Procesor: Intel Core i5-4570 3.20GHz
RAM: 8GB
Rychlost internetového připojení: 40MB/s

14.3 Výsledky

Výsledná data byla zaznamenána do dvou různých forem, a to tabulek (tabulky 14.1 a 14.2) pro možnost detailního vyčtení informace a grafů (obrázky 14.1, 14.2 a 14.3).
Již z tabulek je možné vidět, že co se rychlostí načtení dat ze serveru týče, všechny tři prohlížeče podávají podobné výsledky (v rozmezí od 0.2 sekundy pro malá data do 1.8 sekundy pro ty největší). V grafech porovnávající rychlosti načítání (obrázek 14.1) vidíme, že pro velká data s 50% pokrytím spojů si Firefox vede marginálně hůře, kdežto u 95% pokrytí patří tento lehký propad prohlížeči Internet Explorer 11. Průměrově si však všechny tři prohlížeče v úkolu vedou téměř identicky.

Počet sítí	Doba přenosu [ms]			Doba zpracování [ms]			Celkový čas [ms]		
	Firefox	Chrome	IE11	Firefox	Chrome	IE11	Firefox	Chrome	IE11
30	284	197	178	185	159	410	469	356	588
100	335	357	329	200	156	1419	535	513	1748
200	510	520	496	409	339	3010	919	859	3506
300	625	666	677	535	612	4659	1160	1278	5336
400	807	814	827	758	785	6347	1565	1599	7174
500	1132	1055	1065	986	1042	8534	2118	2097	9599
600	1300	1270	1256	1179	1484	10940	2479	2754	12196

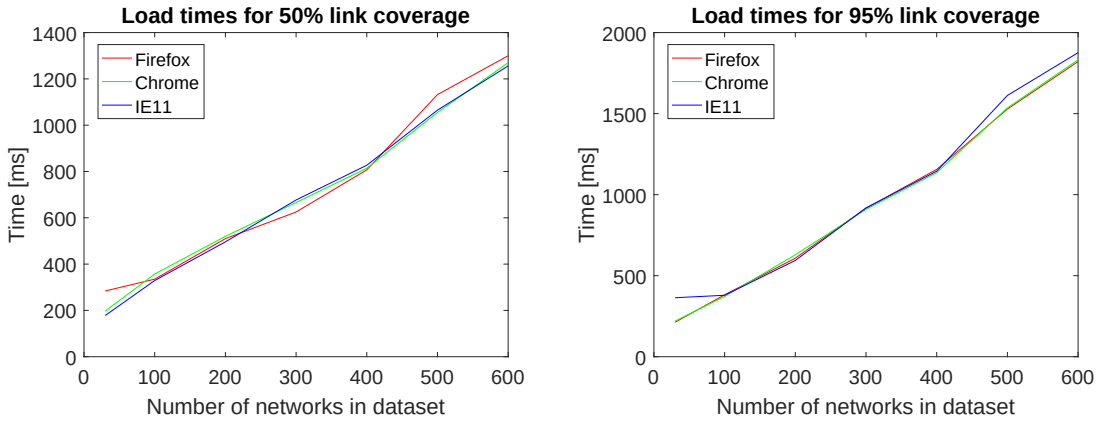
Tabulka 14.1: Výsledky výkonnostních testů pro 50% pokrytí spojů

Počet sítí	Doba přenosu [ms]			Doba zpracování [ms]			Celkový čas [ms]		
	Firefox	Chrome	IE11	Firefox	Chrome	IE11	Firefox	Chrome	IE11
30	213	220	364	163	167	919	376	387	1283
100	382	372	379	185	189	1424	567	561	1803
200	608	629	594	401	347	2995	1009	976	3589
300	916	909	918	584	586	5156	1500	1495	6074
400	1156	1136	1145	715	811	6887	1871	1947	8032
500	1528	1535	1612	1066	1179	9019	2594	2714	10631
600	1821	1832	1876	1282	1574	11452	3103	3406	13328

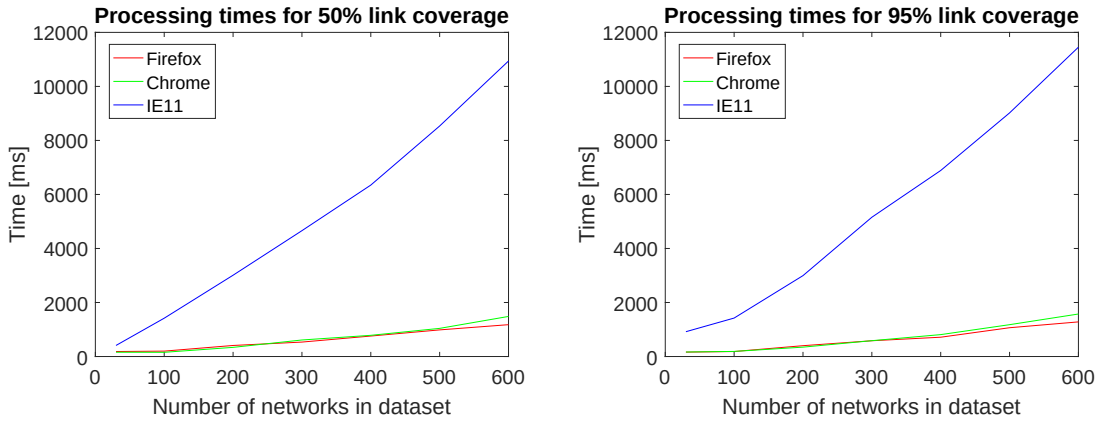
Tabulka 14.2: Výsledky výkonnostních testů pro 95% pokrytí spojů

Podstatně zajímavější se hodnoty stávají v části druhé, kdy dochází ke zpracovávání takto získaných dat javascriptem na straně prohlížeče. Z tabulek i grafů (obrázek 14.2)lze sledovat, že přestože výkony prohlížečů Firefox a Chrome zůstávají rovnocenné, třetí z použitých prohlížečů začíná v této části značně zaostávat a může být pro velké množství informací až o téměř 900% pomalejší než jeho konkurence. Tyto hodnoty již nejsou zanedbatelné, jelikož čekání pro překreslení dat je v této aplikaci již znatelné (zatímco Firefox a Chrome jsou schopny velmi velká data načíst a zpracovat v rozmezí 3 vteřin, IE11 pro tento úkon může vyžadovat více než vteřin 13).

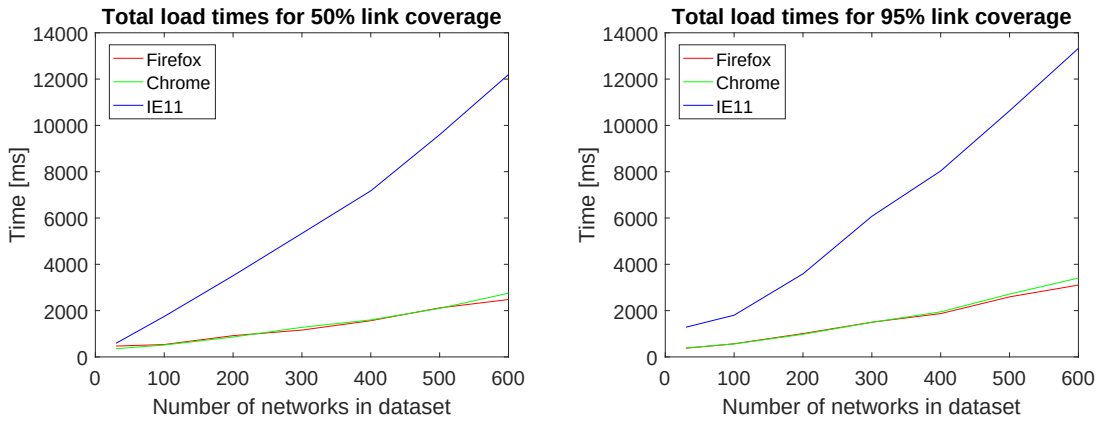
Je tedy zřejmé, že uživatelé získají mnohem příjemnějšího zážitku z využití zde implementovaného nástroje, pokud k prohlížení využijí jeden z populárních moderních internetových prohlížečů namísto vývojáři často negativně přijímaného nativního prohlížeče systému Windows.



Obrázek 14.1: Časy načítání (stažení dat) pro 50% a 95% pokrytí spojů



Obrázek 14.2: Časy zpracování pro 50% a 95% pokrytí spojů



Obrázek 14.3: Celkové časy načtení pro 50% a 95% pokrytí spojů

14.4 Závěry testů

V této kapitole jsme aplikaci otestovali na malých až středně velkých datech (velikost souboru obsahující informace o 600 sítích se pohybovala na hranici 15MB) a zjistili, že přestože jsou implementaci schopny používat všechny tři otestované prohlížeče, uživateli je velmi doporučeno použít moderního webového prohlížeče jako je Mozilla Firefox nebo Google Chrome, které v ohledu zpracování dat podávají až o 900% lepší výsledky než-li výchozí prohlížeč systému Windows.

Z pohledu uživatelského pocitu je také nutno říci, že ačkoli jsou měřené časy prohlížečů Firefox a Chrome téměř identické, může použití na prohlížeči Chrome působit rychleji a plynuleji. Získané údaje však dokazují, že tento pocit je subjektivní a výkon aplikace v obou prohlížečích je v podstatě stejný.

Co se výkonu aplikace samotné týče, pokud se uživatel rozhodne použít doporučený webový prohlížeč, pohybují se časy načtení a vykreslení data setů mezi 0.3 až 3 sekundami, což je pro zamýšlené užití přijatelná časová hodnota. Také je třeba mít na paměti že více než polovina z těchto časů je zabrána stažením datového souboru ze serveru, doba provedení požadavku tedy také přímo závisí na rychlosti datového spojení.

Závěr

O práci

V této práci byla rozebrána problematika vizualizace toku dat v počítačových sítích a to zejména sítích virtuálních nacházejících se uvnitř cloudu. Podívali jsme se na to, jaké způsoby vykreslení této informace existují a jaká řešení jsou použita v produkčních aplikacích, které byly za tímto nebo velmi podobným účelem vytvořeny.
Na základě takto zanalyzovaných informací jsme si vybrali dvě pro naše účely nejvhodněji vypadající vizualizační techniky, kterými byly kruhové zobrazení a maticová heatmapa a vytvořili prototypy navržené pro předpokládaný charakter dat v reálném cloudu.

Přestože se kruhová vizualizace v počátku projektu jevila jako ideální řešení, po dokončení prototypu a vyzkoušení skutečného chování bylo raději přistoupeno k alternativě heatmapy za účelem eliminace omezení počtu dat, které je možné v jednu chvíli zobrazit a vylepšení přehlednosti komunikované informace.

V dalších krocích textu byly identifikovány jednotlivé funkční součásti výsledné aplikace a ty navrženy takovým způsobem, aby zaručovaly vysoký výkon, jednoduchost a především možnost značného rozšíření v budoucích projektech.
Kupříkladu výsledná aplikace nenabízí komplexní práci se systémem uživatelů a uživatelskými nastaveními, jelikož tato funkce nebyla cílem projektu, je nicméně připravena funkční kostra tohoto systému na které lze jednoduchým způsobem takovýto systém vytvořit a do aplikace zaintegrovat.

Součástí návrhu bylo také rozsáhlé zkoumání a testování metod možných k použití pro sběr dat z reálného prostředí cloudu k vizualizaci ve zde vytvořené aplikaci a sepsání doporučeného postupu při tvorbě takovéhoto nástroje (implementace samotná nebyla součástí práce z důvodu nutné stability řešení pro

nasazení v reálném prostředí cloudu a časových prostředků, které by taková implementace a testování vyžadovaly).

Po návrhu všech modulů byly tyto implementovány a otestovány na datovém rozsahu 30 až 600 virtuálních sítí vytvořených za pomocí generátoru dummy dat jehož výstup simuluje charakteristiku reálných dat v námi zkoumaném cloudu.

Některá možná vylepšení

Aplikace byla záměrně navržena takovým způsobem, aby bylo možné jednoduše vytvářet nové funkce a vylepšení, potenciál pro další práci na projektu je tedy velký. Mimo jiné se jedná například o

- *Implementace podpory velmi velkých souborů*
  Diskutováno v kapitole testování, současná forma implementace trpí omezením zde použité databáze MongoDB pro maximální velikost jednoho ukládaného souboru. Tato velikost je dostačující pro uložení dat o přinejmenším 600 sítích, v budoucnu však může nastat případ kdy bude požadována kapacita ještě vyšší.
  V takovém případě stačí pouze upravit databázové PHP soubory MongoDB (využívající systém Doctrine) a zakomponovat mechanismus pro rozdělování velkých souborů do několika menších.

- *Filtrování podle protokolů a další dodatečné filtry*
  Množství filtrů, které lze do takovéto aplikace přidat je opravdu omezené jen potřebami a požadavky uživatele, existuje jich tedy velmi velké množství. Filtrovací mechanismy v aplikaci použité patří mezi ty nejzákladnější a mezi těmi které se nabízí pro další implementaci je například schopnost aplikace filtrovat zobrazovaná data podle protokolů (současná verze zobrazuje agregaci napříč všemi protokoly).
  Také je možné vytvořit filtr umožňující odstranění sítí či adres jejichž celková zátěž nepřesahuje danou minimální hodnotu (spodní hranice v aktuální verzi je pevně nastavená na nulu).

- *Uživatelský systém*
  Velké množství dodatečných funkcí nabízí systém správy uživatelů. V aplikaci je již například připraven (avšak nepoužit) systém uživatelských rolí, díky kterým by bylo možné uživatele omezit v tom jaké data sety (a které sítě v nich) se jim zobrazí a jaké akce jim s nimi bude povoleno dělat.

- *Podpora vlastních mezních hodnot, škál a dalších preferencí*
  Aplikace v současné chvíli dovoluje používat globální a lokální škálový systém, to však nemusí být dostatečné pro všechny účely a uživatel si

například může přát nastavit pevnou hodnotu maximálního toku, vůči kterému budou intenzity barev vztaženy.
To samé lze říci o množství dalších uživatelských preferencí, které lze do aplikace zakomponovat. Funkční kostra pro toto rozšíření je již také připravena na obou stranách aplikace, ačkoli jediná poskytnutá možnost je v tuto chvíli aktivace či deaktivace automatického načtení nejnovějších dat při startu stránky.

Literatura

[1] Li, Bingdong, Jeff Springer, George Bebis, and Mehmet Hadi Gunes. "A Survey of Network Flow Applications."Journal of Network and Computer Applications 36, no. 2 (2013): 567-81. doi:10.1016/j.jnca.2012.12.020.

[2] Tamassia, R. Handbook of graph drawing and visualization. Boca Raton: CRC Press Taylor & Francis Group. 2016.

[3] Corneliu Sugar. *US airline routes graph.*[přístup 20.května 2018]. Dostupné z: http://bl.ocks.org/upphiminn/6515478

[4] Tufte, Edward R. *The Visual Display of Quantitative Information.* Cheshire, CT: Graphics Press, page 25, 2001.

[5] Zavou, Angeliki, et al. "Cloudopsy: An Autopsy of Data Flows in the Cloud." Lecture Notes in Computer Science Human Aspects of Information Security, Privacy, and Trust, 2013, pp. 366–375., doi:10.1007/978-3-642-39345-7_39.

[6] www.visualcinnamon.com/.*Hacking a chord diagram to visualize a flow.*[přístup 20. května 2018]. Dostupné z: https://www.visualcinnamon.com/2015/08/stretched-chord.html

[7] Khomtchouk BB, Hennessy JR, Wahlestedt C (2017) shinyheatmap: Ultra fast low memory heatmap web interface for big data genomics. PLoS ONE 12(5): e0176334. https://doi.org/10.1371/journal.pone.0176334

[8] www.data-imaginist.com. *Introduction to ggraph: Layouts.*[software].[přístup 20.května 2018].Dostupné z: https://www.data-imaginist.com/2017/ggraph-introduction-layouts/

[9] www.ntop.com. *ntopng.*[software].[přístup 29. prosince 2017].Dostupné z: `https://www.ntop.org/products/traffic-analysis/ntop/`

[10] Caida.*FlowScan.*[software].[přístup 29. prosince 2017].Dostupné z: `https://www.caida.org/tools/utilities/flowscan/`

[11] *NfSen.*[software].[přístup 29. prosince 2017].Dostupné z: `http://nfsen.sourceforge.net/`

[12] *NfDump.*[software].[přístup 29. prosince 2017].Dostupné z: `https://github.com/phaag/nfdump`

[13] *EtherApe.*[software].[přístup 29. prosince 2017].Dostupné z: `http://etherape.sourceforge.net/`

[14] *FloVis.*[software].[přístup 29. prosince 2017].Dostupné z: `https://web.cs.dal.ca/~sbrooks/projects/NetworkVis/index.html`

[15] Braun, L., Volke, M., Schlamp, J., Bodisco, A. V., Carle, G. (2013). Flowinspector: a framework for visualizing network flow data using current web technologies.*Computing, 96(1)*, 15-26. doi:10.1007/s00607-013-0286-4

[16] Shiravi, H., Shiravi, A., Ghorbani, A. A. (2012).A Survey of Visualization Systems for Network Security.*IEEE Transactions on Visualization and Computer Graphics*, 18(8), 1313-1329. doi:10.1109/tvcg.2011.144

[17] The JavaScript Object Notation (JSON) Data Interchange Format. (n.d.).[software].[přístup 20. května 2018]. Dostupné z: https://tools.ietf.org/html/rfc7159

[18] ECMAScript® Language Specification.[software].[přístup 20. května 2018]. Dostupné z: https://www.ecma-international.org/ecma-262/5.1/

[19] Bostock, M. (n.d.). Data-Driven Documents.[software].[přístup 20. května 2018]. Dostupné z: https://d3js.org/

[20] Castillo, P. N. Mastering D3.js: Bring your data to life by creating and deploying complex data visualizations with D3.js. Birmingham: Packt Pub. 2014.

[21] Apache.*Apache2.*[software].[přístup 20. května 2018]. Dostupné z: httpd.apache.org

[22] NGINX Inc.*Nginx.*[software].[přístup 20. května 2018]. Dostupné z: https://www.nginx.com/

[23] rubyonrails.org.*Ruby on Rails.*[software].[přístup 20. května 2018]. Dostupné z: https://rubyonrails.org/

[24] Sensio Labs.*Symfony.*[software].[přístup 20. května 2018]. Dostupné z: http://symfony.com/

[25] jquery.com.*jQuery.*[software].[přístup 20. května 2018]. Dostupné z: https://jquery.com/

[26] Oracle.*MySQL.*[software].[přístup 20. května 2018]. Dostupné z: https://www.mysql.com/

[27] PostreSQL/.*PostgreSQL.*[software].[přístup 20. května 2018]. https://www.postgresql.org/

[28] Apache.*Cassandra.*[software].[přístup 20. května 2018]. Dostupné z: https://www.postgresql.org/

[29] Apache.*Hadoop.*[software].[přístup 20. května 2018]. Dostupné z: http://hadoop.apache.org/

[30] MongoDB, Inc.*MongoDB.*[software].[přístup 20. května 2018]. Dostupné z: https://www.mongodb.com/

[31] Doctrine-project.org.*Doctrine.*[software].[přístup 20. května 2018]. Dostupné z: https://www.doctrine-project.org/

[32] Sensio Labs.*Twig.*[software].[přístup 20. května 2018]. Dostupné z: https://twig.symfony.com/

[33] Kernel Virtual Machine.*KVM.*[software].[přístup 20. května 2018]. Dostupné z: https://www.linux-kvm.org/page/Main_Page

[34] Saltstack.*SaltMaster.*[software].[přístup 20. května 2018]. Dostupné z: https://saltstack.com/

[35] Linux Foundation Collaborative Project.*Open vSwitch.*[software].[přístup 20. května 2018]. Dostupné z: https://www.openvswitch.org/

[36] Gerard Paul Java.*iptraf.*[software].[přístup 20. května 2018]. Dostupné z: http://iptraf.seul.org/

[37] Cisco.*NetFlow Version 9 Flow-Record Format.*[přístup 20. května 2018]. Dostupné z: https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html

[38] Errata Exist.*Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information.*[přístup 20. května 2018]. Dostupné z: https://www.ecma-international.org/ecma-262/5.1/

[39] sFlow.org.*sFlow.*[software].[přístup 20. května 2018]. Dostupné z: https://sflow.org/developers/specifications.php

[40] nfcapd, nfdump.[software].[přístup 20. května 2018]. Dostupné z: https://github.com/phaag/nfdump

[41] Gregor Aisch. *Chroma.js Color Scale Helper*[software].[přístup 20. května 2018]. Dostupné z: https://gka.github.io/palettes/#diverging|c0=darkred,deeppink,white|c1=white,lightgreen,teal|steps=13|bez0=1|bez1=1|coL0=1|coL1=1

Příloha **A**

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

BSON Binary JSON

CSS Cascading Style Sheets

DOM Document Object Model

HTML HyperText Markup Language

IP Internet Protocol

IPFIX IP Flow Information Export

JSON JavaScript Object Notation

MAC Media Access Control

OS Operating System

RAM Random Access Memory

REST API Representational State Transfer API

SQL Structured Query Language

TCP Transmission Control Protocol

UDP User Datagram Protocol

UI User Interface

VLAN Virtual Local Area Network

VM Virtual Machine