Masarykova univerzita
Fakulta informatiky


# Vizualizace jevů a jejich stavových změn pro projekt KYPO

Bakalářská práce

**Ondřej Bulla**

Brno, jaro 2017

# Prohlášení

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

Ondřej Bulla

**Vedoucí práce:** Mgr. Miroslava Jarešová

# Poděkování

Rád bych poděkoval vedoucí mé práce Mgr. Miroslavě Jarešové, za všechnu pomoc, kterou mi poskytla, za veškerý čas, který mi věnovala, a za velice přátelský přístup. Dále bych chtěl poděkovat Tomáši Plesníkovi, za všechny cenné konzultace a Štěpánu Matalíkovi, Mgr. Bc. Davidu Koňařovi, Mgr. Josefu Buštovi, Bc. Janu Tomáškovi, RNDr. Tomáši Jirsíkovi, Mgr. Ing. Jakubu Čeganovi, Karlu Vašákovi a ještě jednou Tomáši Plesníkovi za to, že si udělali čas na uživatelské testování této práce.

# Shrnutí

Tato práce se zabývá implementací vizualizace jevů a jejich stavových změn pro projekt KYPO. Vizualizace poskytuje uživatelům přehled o událostech probíhajících na jednom uzlu sítě. Některými svými funkcemi přesahuje i do vizualizace síťové topologie.

Při implementaci byl kladen důraz na obecnost využití a škálovatelnost, díky čemuž může být vizualizace aplikována na různé případy užití. Díky snadné konfigurovatelnosti můžou být vizualizována libovolná data stavového charakteru.

Výsledkem této práce je standalone aplikace, která pracuje s daty vytvářenými generátorem dat a mockupem vizualizace síťové topologie. Aplikace byla vytvářena s ohledem na architekturu projektu KYPO, aby byla zjednodušena integrace vizualizace do síťové topologie s reálnými daty. Rozhraní se zdrojem reálných dat nebylo v čase zpracovávání této práce ještě dostupné.

# Klíčová slova

JavaScript, KYPO, D3, vizualizace, služby, síť, SVG, Kybernetický polygon, časová osa, HTML, stavy

# Obsah

# Seznam obrázků

# 1 Úvod

Žijeme v době přebytku informací. Je jich v našem každodenním životě mnohem více, než zvládáme přijímat. Vzniká tedy nový úkol — naučit se informace předávat efektivně, aby byly pro příjemce co nejpřehlednější a co nejjednodušeji pochopitelné.

Jedním z velice efektivních nástrojů na předávání informací je **vizualizace dat**[1]. „V současné informační společnosti (...) je nutné se umět s daty nejen vypořádat a analyzovat je, ale dokázat je i efektivně vizuálně prezentovat, a to ať už je našim cílem jejich objektivní komunikace, nebo argumentace a přesvědčování.“ [2]

Proč ale data prezentovat právě vizuálně? Efektivita tohoto způsobu prezentace dat je založena na faktu, že člověk zrakovým vjemem přijímá více podnětů než všemi ostatními smysly dohromady. „Málo z nás umí rozpoznat vzory mezi řadami čísel, ale dokonce i malé děti umí interpretovat sloupcové diagramy a vyvodit význam z těchto vizuálních reprezentací čísel.“ [3]

Vizualizace je tedy v tomto ohledu výborným pomocníkem. A nemusíme skončit jen u grafů a sloupcových diagramů. V současné době jsou vizualizace často interaktivní a uživatel může sám ovlivnit, jakou část dat si bude prohlížet, co chce zvýraznit, co naopak upozadit, atd. To dělá prezentaci informací ještě srozumitelnější.

Cílem této práce je vytvořit vizualizaci jevů a jejich stavových změn pro projekt **KYPO**[2] (Kybernetický polygon), který se zabývá výzkumem, vývojem a sestavením unikátního prostředí pro analýzu hrozeb ohrožujících bezpečnost kritických informačních infrastruktur, a který vzniká na Fakultně informatiky Masarykovy univerzity v Brně.

Tato vizualizace je naprogramovaná tak, aby byla schopna vizualizovat jakékoliv data stavového charakteru. KYPO je rozsáhlý projekt, kde by tato vizualizace mohla mít mnohá využití, proto bylo potřeba ji vytvořit co nejobecněji. Konkrétní využití vizualizace se tedy snadno změní dodáním jiných vstupních dat. V mém případě jsem jako vstupní data použil údaje získané měřením dostupnosti služeb na síťovém uzlu.

---

1. Běžným postupům při vytváření vizualizací se věnuje například kniha Visual Insights [1].
2. `http://www.kypo.cz`

Vizualizace je interaktivní, díky čemuž může uživatel ovlivnit, na co se chce při sledování dat zaměřit. Součástí práce je také mockup vizualizace síťové topologie a generátor dat simulující měření stavů služeb, protože ještě není zajištěno dodávání dat ze skutečných měření.

## 1.1 Motivace

Důvodem vzniku této vizualizace je potřeba zjednodušit přehled nad jevy v počítačové síti.

V projektu KYPO bude tato vizualizace využita např. při cvičeních Cyber Czech[3]. Díky přehledu nad celou sítí mohou rozhodčí (tzv. White Team) snadno udělovat týmům při cvičení body (resp. strhávat body, pokud jsou nějaké služby na uzlech nedostupné).

Dalším využitím může být například demonstrace jevů, které v síti probíhají během napadení sítě.

## 1.2 Přehled jednotlivých kapitol

Úvodní kapitoly obsahují shrnutí mého pracovního postupu a stručný popis projektu KYPO. Následuje kapitola týkající se vizualizace, která tenhle problém již řešila, a důvodů, proč se od jejího využití nakonec upustilo. Hned po ní je kapitola stručně popisující nový návrh řešení této vizualizace.

Po tomto stručném uvedení do problematiky následují obsáhlejší kapitoly popisující funkcionalitu výsledné práce a vybrané části implementace, včetně popisu použitých technologií.

V závěrečných kapitolách je popsáno uživatelské testování, dále co je nutné udělat pro to, aby byla vizualizace úspěšně integrována do projektu KYPO a možnosti rozšíření funkcionality.

3. `https://csirt.muni.cz/projects/cyber-czech`

# 2 Průběh práce

V úvodu práce na této vizualizaci jsem se scházel s týmem KyVi, který je pověřený tvorbou vizualizací pro projekt KYPO a jehož vedoucím je RNDr. Radek Ošlejšek Ph.D. Na těchto schůzkách jsem byl stručně seznámen s KYPO a s technologiemi, které se aktuálně při vývoji vizualizací v projektu používají. Vzhledem k webovému charakteru systému se jednalo konkrétně o HTML, SVG, CSS, JavaScript a převážně jeho knihovnu **D3.js**. Aby mohla být vizualizace následně snadno integrovaná do KYPO, bylo potřeba ji vyvíjet pomocí stejných technologií. Jelikož jsem s nimi měl pouze omezené zkušenosti, prvním krokem pro mě bylo seznámit se s nimi a naučit se je používat.

Nejvíce jsem při práci využíval knihovnu D3.js, takže jsem své studium zaměřil převážně tímto směrem. Vycházel jsem z knihy Scotta Murrayho **Interactive Data Visualization for the Web** [3]. Ta je převážně učebnicí knihovny D3.js, ale v úvodních kapitolách se věnuje také HTML, SVG, CSS a JavaScriptu, takže pokryla všechny mé potřeby.

Již před četbou této učebnice jsem se seznámil s návrhem řešení vizualizace, který vytvořila Mgr. Miroslava Jarešová. Při čtení knihy jsem tedy již myslel na to, které technologie v ní uvedené by se mi mohly při práci hodit, a zaměřil jsem na ně více pozornosti.

Po nastudování programovacích jazyků a seznámení se s návrhem jsem začal se samotnou implementací. Protože zatím není vyřešeno dodávání dat pro vizualizaci, bylo nejprve potřeba vytvořit **generátor dat** simulující měření stavů služeb. Seznámil jsem se tedy se strukturou dat, používaných v praxi, a vytvořil jsem generátor, který v pravidelných časových intervalech generuje data, která jsou pak vizualizována. Tento generátor byl později ještě upravován, aby generovaná data co nejvíce odpovídala reálným datům. Díky tomu bylo možné odhalit a napravit nedostatky vizualizace, vyplývající ze struktury dat, již při práci se simulovanými daty.

Protože je výsledná vizualizace úzce spojena i s **vizualizací síťové topologie**, bylo potřeba i tu nějakým způsobem zakomponovat do mé práce. Ve vizualizaci síťové topologie byla potřeba především možnost zvolení si uzlu, pro který chce uživatel zobrazit vizualizaci jevů a jejich stavových změn a dále pak některé doplňkové funkce,

např. zvýrazňování uzlů, volba sekudárních uzlů nebo upozornění na změnu stavu sledované služby. Nebyla potřeba zobrazovat celou mapu topologie sítě, včetně spojení mezi jednotlivými uzlu.

Z těchto důvodů jsme se rozhodli, že si, stejně jako u dodávaných dat, vystačíme s mockupem. Ten jsem tedy také musel vytvořit předtím, než jsem mohl začít se samotnou prací na vizualizaci jevů a jejich stavových změn. Vizualizace síťové topologie je tedy oproti té, která bude použita v praxi, značně zjednodušená. Kvůli absenci mapy topologie, se vlastně ani nejedná o vizualizaci topologie sítě, ale spíše o seznam uzlů, u kterých je možné vizualizaci jevů zobrazit.

Po vytvoření tohoto mockupu už jsem plně pracoval na vizualizaci, která je hlavním výstupem této práce. Po celou dobu jsem novinky v mé práci konzultoval nejen s vedoucí práce Mgr. Miroslavou Jarešovou, ale i s Tomášem Plesníkem, který je zaměstnancem CSIRT-MU[1] a je jedním z lidí, kteří budou výsledek mé práce používat v praxi. Mimo to byla má práce v méně častých intervalech také prezentovaná týmu KiVi.

Po dokončení implementace vizualizace jevů jsem výsledný program podrobil **uživatelskému testování**. To ověřilo použitelnost vizualizace a přineslo mnoho návrhů na upravení nebo přidání funkcionality. Některé z těchto návrhů jsem poté do implementace zapracoval a zbylé jsem popsal v kapitole 9 – Uživatelské testování.

1. `https://csirt.muni.cz/`

# 3 Kybernetický polygon

Projekt Kybernetický polygon (dále jen KYPO)[1] se zabývá vytvořením jedinečného **síťového prostředí**, které může sloužit pro výzkum a vývoj metod na ochranu proti útokům na kritické informační infrastruktury.

Díky tomuto prostředí se mohou uskutečňovat bezpečnostní experimenty a cvičení bezpečnostních týmů starajících se o správný chod výše zmíněných infrastruktur. Cvičení jsou realizovaná v KYPO laboratoři, která se nachází v budově Fakulty informatiky Masarykovy univerzity.

Projekt je finančně podporován Ministerstvem vnitra ČR a je vyvíjen Ústavem výpočetní techniky ve spolupráci s Fakultou informatiky Masarykovy univerzity. [4]

## 3.1 Možnosti využití

Prostředí Kybernetického polygonu poskytuje tři základní možnosti využití. První z nich představuje vzdělávání bezpečnostních odborníků a členů bezpečnostních týmů. Druhým je testování, vyhodnocování a demonstrace nových způsobů detekce a obrany proti útokům vůči kritické informační infrastruktuře. Třetí pak představuje prostředí KYPO jako místo pro realizaci forenzních analýz a bezpečnostních experimentů nad počítačovými sítěmi. [5]

### 3.1.1 Bezpečnostní školení a cvičení

Díky projektu KYPO je možné pořádat školení a cvičení uživatelů, administrátorů i bezpečnostních týmů počítačových systémů. Taková cvičení jsou žádoucí především u významných informačních systémů, jako jsou např. systémy Armády ČR, Českého telekomunikačního úřadu, Ministerstva obrany, Ministerstva vnitra, Policie ČR nebo Úřadu vlády. [6]

1. `http://www.kypo.cz`

Hlavní výhodou cvičení v rámci projektu KYPO je vysoká míra interaktivity. Projekt umožňuje přesně namodelovat infrastrukturu sítě, se kterou školené týmy reálně pracují.

Pomocí monitoringu aktivity uživatelů je možné předat jim během cvičení cennou zpětnou vazbu.

Mezi realizovaná cvičení patří cvičení **Cyber Czech** každoročně vytvořené ve spolupráci s Národním bezpečnostním úřadem, ale také cvičení realizovaná pro komerční nebo akademické partnery. [5]

### 3.1.2 Výzkum ochrany proti kybernetickým útokům

Při vytváření Kybernetického polygonu se myslelo nejen na trénink a školení, ale i na výzkum. V KYPO je tedy možné vyvíjet, testovat a demonstrovat nové metody detekce síťových útoků a obrany proti nim.

KYPO má **vestavěnou měřící infrastrukturu**, která umožňuje zaznamenat veškerý síťový provoz, který je pak možné zpětně podrobně analyzovat. [5]

### 3.1.3 Forenzní analýza a síťové simulace

V KYPO je možné do virtuálního prostředí vložit předpřipravené obrazy strojů včetně mobilních zařízení a měnit jejich vlastnosti (síťové připojení, kapacita linek, atd.). Vestavěná monitorovací infrastruktura umožňuje sledovat stav strojů a sítě a následné forenzní zkoumání kompromitovaných strojů. Virtuální prostředí poskytuje možnost napadený stroj spouštět opakovaně a analyzovat jeho chování. [5]

## 3.2 Vizualizace v KYPO

Pro projekt KYPO je vyvíjena řada vizualizací, které mají za cíl zlepšit uživatelův přehled o tom, co se v síti děje. Vizualizace jevů a jejich stavových změn funkčně navazuje na již existující vizualizaci síťové topologie.

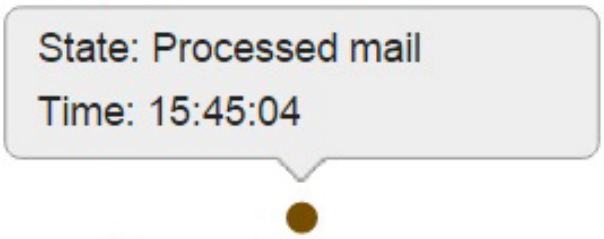
# 4 Předchozí řešení vizualizace jevů

Vizualizace podobného charakteru, se specifikací na případ užití přehledu stavů emailů, byla pro projekt KYPO již dříve vyvinuta. Vypracoval ji Mgr. Tomáš Bobek v rámci své diplomové práce. [7]

Jeho řešení se ale zaměřovalo příliš konkrétně na jeden případ užití a bylo potřeba vytvořit vizualizaci, která by byla lépe škálovatelná, a proto byl vypracován nový návrh, realizovaný v této práci.

Hlavním rozdílem mezi těmito dvěma vizualizacemi je ten, že vizualizace Mgr. Tomáše Bobka nepracovala s žádnou časovou osou nebo grafem. Změna stavu jednotlivých služeb byla vykreslována pouze **barevnými kolečky**, která se řadila za sebe v pořadí, v jakém se různé události (e-maily) objevovaly (obr. 4.1). Jeden řádek pak reprezentuje jednu kategorii událostí a barva kolečka definuje stav, ve kterém se událost právě nachází. Čas změny stavu se dal zjistit pouze najetím myší na kolečko (obr. 4.2) a historie stavů dané události není zjistitelná vůbec. Barvu, kterou se má konkrétní stav události vykreslovat, mohl uživatel nastavit dle libosti.


Obrázek 4.1: Předchozí řešení vizualizace jevů a jejich stavových změn [7]




Obrázek 4.2: Zobrazení informací o službě pomocí najetí myší [7]

Důvodem tohoto zjednodušení bylo, že vizualizace jevů měla být zahrnuta přímo do vizualizace síťové topologie a bylo tedy potřeba, aby nezabírala moc místa, protože vizualizace síťové topologie by se pak stala nepřehlednou. Vizualizace jevů se totiž měla zobrazovat přímo vedle uzlu. Ve chvíli, kdy je topologie sítě rozsáhlá a obsahuje mnoho uzlů, je potřeba šetřit místem a vizualizaci jevů tedy zobrazovat co nejúsporněji. To ale výrazně snižovalo její vypovídající hodnotu.

Toto řešení obsahuje tyto hlavní nevýhody:

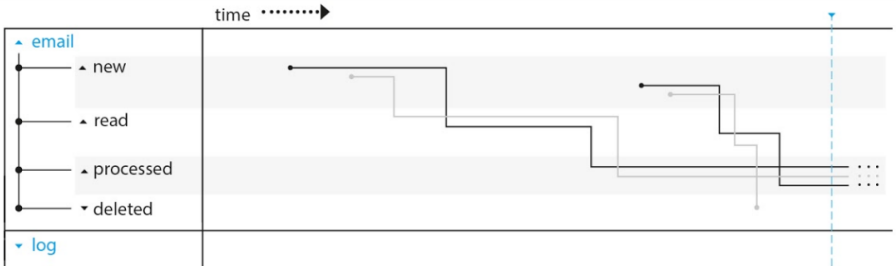
- Neexistuje možnost zobrazení historie stavů dané události.
- Řešení není škálovatelné a v případě, že je potřeba vizualizovat mnoho kategorií (řádků) nebo událostí (sloupců), může vizualizace zabírat příliš mnoho místa.
- Není možné zjistit, kdy se událost na uzlu vyskytla poprvé.

Tohle řešení bylo tedy opuštěno a vznikl návrh nové vizualizace, která by byla zobrazována ve vlastním portletu, a měla tedy možnost být tak rozsáhlou (a přehlednou), jak je potřeba.

# 5 Návrh vizualizace

Při mé práci jsem postupoval podle již vytvořeného návrhu vizualizace, jehož autorkou je Mgr. Miroslava Jarešová.

Mockup, který vytvořila (obr. 5.1), obsahoval základní představu o tom, jak má vizualizace vypadat a jakou funkcionalitu bude obsahovat.




Obrázek 5.1: Návrh vizualizace [8]

Hlavním rozdílem oproti předchozímu řešení je to, že vizualizace je zobrazena v samostatném portletu – odděleně od vizualizace síťové topologie. Díky tomu má dostatek prostoru a může tím pádem přehledněji popsat, co se na vybraném uzlu děje.

Zatímco předchozí vizualizace obsahovala pouze barevné kruhy, reprezentující stav, ve kterém se sledovaný jev nachází, nový návrh vizualizace obsahuje přehlednější **oblast grafu**, kde vodorovná osa reprezentuje průběh času[1] a svislá osa obsahuje barevně oddělené řádky, z nich každý reprezentuje jeden stav, ve kterém se sledovaný jev může nacházet. Jev samotný se pak do grafu zanáší pomocí lomené čáry, přičemž je možné sledovat v grafu několik jevů najednou a mít tedy možnost okamžitě porovnat jejich chování.

Některé aspekty vzhledu byly v průběhu práce změněny. Vznikly také návrhy nové funkcionality, které v původním návrhu nebyly obsaženy. Tyto změny většinou vyplývaly z konzultací jak s Mgr. Miroslavou Jarešovou, tak s Tomášem Plesníkem.

---

1. Přehled nejčastějších postupů při vizualizaci dat časového charakteru lze nalézt v knize Visualization of Time-Oriented Data [9].

Zadané funkční požadavky:

- Vizualizace bude obsahovat **časový bod**, kterým bude uživatel moci hýbat.
- Oblast grafu bude možné **posouvat** doleva i doprava.
- Oblast grafu bude možné **přibližovat** a **oddalovat**.
- Vizualizaci je možné sledovat ve dvou módech. Implicitně se spouští v **heart-beat** módu, což znamená, že časový okamžik je *„ukotven“* v současnosti a graf tedy postupně *„ubíhá“* doleva. **Explorační mód** umožňuje uživateli průzkum minulosti grafu pomocí pohybu časového bodu nebo celého grafu.
- Ve vizualizaci síťové topologie bude možné vykreslit **upozornění** na to, že se daná služba ocitla v daném stavu. Tato upozornění bude moci uživatel nastavit pomocí konfiguračního dialogu. Nastavení bude možné trvale uložit.

  Budou implementovány dva způsoby upozornění - tzv. **upozornění bublinou** a **upozornění barevným zvýrazněním rohu uzlu**.

  Při exploraci grafu do minulosti budou zobrazovaná upozornění odpovídat okamžiku, ve kterém se nachází časový bod.
- Jednotlivé služby se do grafu zaznamenávají pomocí **lomených čar**. Ty se mohou zobrazovat buď odděleně, nebo **agregovaně** v rámci stavů - pak šířka čáry reprezentuje počet služeb, které se právě v daném stavu vyskytují.

  Při najetí myší na čáru se zobrazí **popisek** dané služby.

  Čáry reprezentující jednotlivé služby bude možné **zvýraznit**.
- Jednotlivé služby budou rozděleny do **kategorií**. Služby každé kategorie budou vykresleny v samostatném grafu. Kategorie bude také možné skrývat.
- Do vizualizace bude možné zahrnout i **sekundární uzly**.

K těmto funkčním požadavkům se během implementace přidaly ještě požadavky na **přehled** nad grafem, **barevná pozadí pod grafem**, včetně možnosti nastavení těchto barev a možnost rozdělení služeb i do **podkategorií**.

Zadané nefunkční požadavky:

- Vizualizace bude vytvořena pomocí knihovny **D3.js** jazyka JavaScript.
- Při implementaci bude brána v potaz architektura portálu KYPO a implementace bude řešena tak, aby byla následná integrace do KYPO co nejjednodušší.

Původní návrh, podle kterého jsem při implementaci postupoval, je součástí elektronického archivu práce. Jedná se o soubor specifikace.pdf.

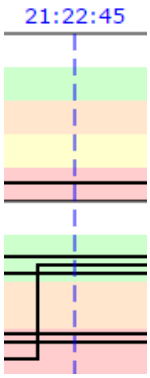
# 6 Popis funkcionality

V této kapitole jsou popsány naimplementované funkce – jejich význam, obsluha a důvod jejich vzniku. Součástí této kapitoly jsou také obrázky zobrazující detaily vizualizace. Představu o vzhledu vizualizace jako celku můžete získat v příloze A (screenshoty).

## 6.1 Manipulace s grafem

### 6.1.1 Časový bod

Časový bod je v grafu reprezentován svislou modrou přerušovanou čarou s časovým údajem nad ní (obr. 6.1). Časový údaj nad čárou vždy sděluje čas odpovídající místu, kde se čára právě nachází.

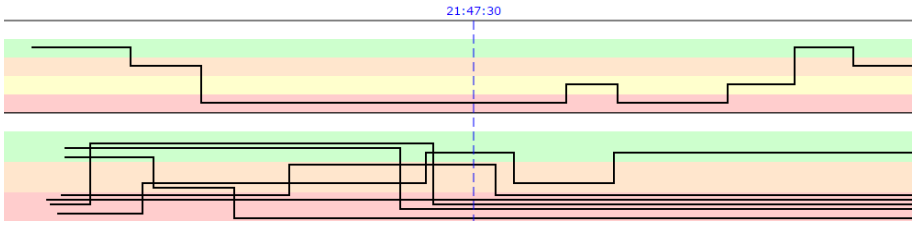



Obrázek 6.1: Časový bod

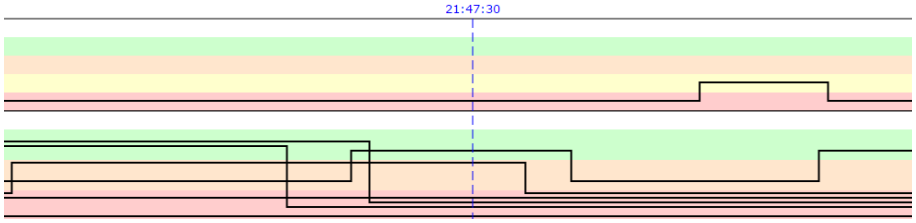
Časový bod lze chytnout myší a následně s ním pohybovat. Časový popisek se pak automaticky aktualizuje podle toho, v jakém místě grafu se čára právě nachází. Takto se dá snadno zjistit přesný čas daného místa v grafu.

### 6.1.2 Přibližování/oddalování

Graf je možné přiblížit nebo oddálit a to pomocí kolečka myši (obr. 6.2 a 6.3). Místo, kde se právě nachází časový bod, je pak v grafu ukotveno na místě a zbytek grafu se upravuje relativně k tomuto místu.




Obrázek 6.2: Oddálený graf




Obrázek 6.3: Přiblížený graf

### 6.1.3 Posouvání

Graf je možné posouvat doleva a doprava a to následujícími způsoby:

- Chycení pozadí grafu myší a pohybem myši.
- Držením mezerníku a pohybem myši.
- Chycením časového bodu myší a jeho pohybem až k okraji oblasti grafu.
- Chycením oranžového obdélníku v přehledu myší a pohybem myši.

Grafem lze posouvat doleva jen po okamžik, kdy se v grafu vyskytuje první údaj a doprava jen po okamžik odpovídající současnosti.

### 6.1.4 Přehled

Přehled (obr. 6.4) je umístěný nad grafem. Uživatel se díky němu dozví, jak velkou část naměřených údajů právě sleduje.


Obrázek 6.4: Přehled

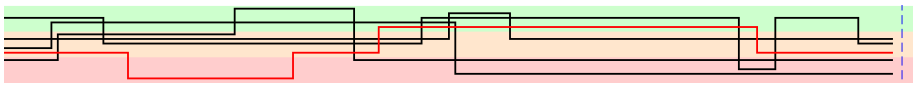
Šedý obdélník na pozadí vyjadřuje veškerý čas, kdy byla měření prováděna. Čas prvního měření je uveden vlevo od obdélníku a vpravo od obdélníku je uveden aktuální čas. Oranžový obdélník reprezentuje časový úsek, který je aktuálně zobrazen v grafu.

Oranžovým obdélníkem lze pomocí myši pohybovat, což způsobí posouvání celého grafu.

Také lze pomocí myši pohybovat černými okraji oranžového obdélníku. To způsobí změnu jeho velikosti a tedy i změnu velikosti zobrazovaného časového úseku.

### 6.1.5 Zvýrazňování a popis služeb

Protože jsou všechny měřené služby vykreslovány do grafu stejnou barvou, může být pro uživatele obtížné se orientovat v tom, která čára patří které službě. Aby byla orientace usnadněna, obsahuje vizualizace možnost zvýraznění vybrané služby. Daná služba je pak vykreslena červenou čarou (obr. 6.5).

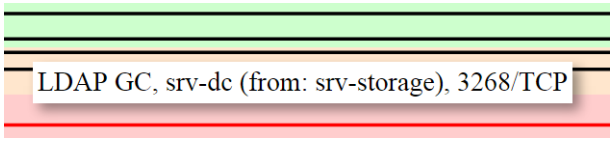


Obrázek 6.5: Zvýraznění služby

Zvýraznění může být dočasné (najetí myší na čáru) nebo trvalé (kliknutí na čáru). Trvale zvýrazněná může být pouze jedna služba z kategorie (podkategorie). Pokud je kliknuto na jinou službu ze stejné kategorie, zvýraznění předchozí služby se zruší. Zvýraznění služby lze také zrušit kliknutím do oblasti grafu mimo jakoukoliv službu.

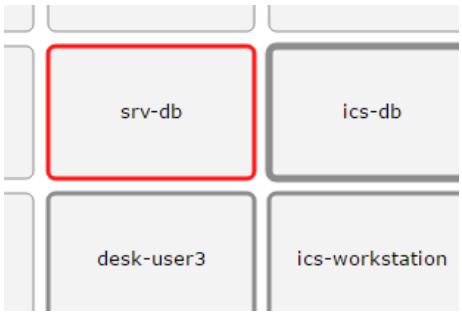
Při najetí myší na čáru se kromě dočasného zvýraznění čáry objeví také popis dané služby (obr. 6.6). V mé práci tento popis obsahuje název služby, uzel, na kterém je měřena, uzel, ze kterého je měřena, port, přes který je měřena, a protokol, pomocí kterého je měřena. Výjimkou je služba overall, jejíž popisek obsahuje pouze název služby a uzel, na kterém je měřena.

Formát textu popisujícího službu lze snadno změnit přeprogramováním funkce getServiceLabel(datum). Parametr datum je objekt obsahující informace o měřené službě[1].




Obrázek 6.6: Popis služby

Při najetí myší na čáru se také ve vizualizaci síťové topologie zvýrazní uzel, na kterém se daná služba měří (obr. 6.7). To je užitečné ve chvíli, kdy má uživatel ve vizualizaci zahrnuto více uzlů. Takto uživatel snadno zjistí, kterému z uzlů služba náleží.

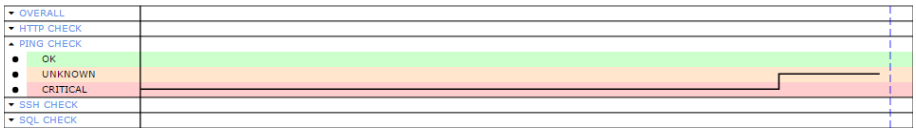



Obrázek 6.7: Zvýraznění uzlu

1. viz 10.1.1 Formát dat

### 6.1.6 Skrývání kategorií

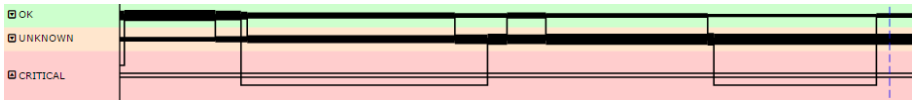
Kategorie (a podkategorie) mohou být skryty (obr. 6.8) pomocí kliknutí na šipku vedle jejich názvu. Po dalším kliku se kategorie opět zobrazí.




Obrázek 6.8: Graf s několika skrytými kategoriemi

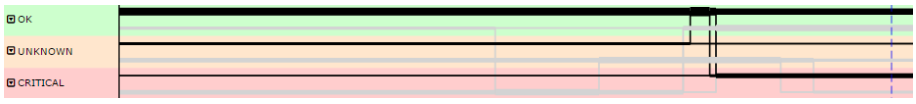
### 6.1.7 Agregace čar

U kategorií, které obsahují alespoň dvě měřené služby, je možnost čáry reprezentující služby v grafu agregovat, pomocí kliknutí na ikonku vedle názvu stavu, jehož čáry agregovat chceme. Z několika samostatných čar se tak stane jedna (obr. 6.9). Tloušťka této čáry pak odpovídá počtu služeb, které se v daném čase ve vybraném stavu nachází.




Obrázek 6.9: Graf s agregací stavů OK a UNKNOWN

Pokud jsou do vizualizace zahrnuty i sekundární uzly, pak je zobrazovaná jedna agregovaná čára pro každý sekundární uzel (obr. 6.10).




Obrázek 6.10: Graf s agregací všech stavů při vizualizování jevů jednoho primárního a jednoho sekundárního uzlu

Tato funkce usnadní orientaci v grafu např. ve chvíli, kdy se uživatel chce zaměřit jen na služby, které jsou aktuálně ve stavu CRITICAL. Pak může agregovat zobrazení stavů OK a UNKNOWN a bude sledovat jen poměrové zastoupení služeb v těchto stavech. Služby, nacházející se ve stavu CRITICAL, ale stále uvidí se všemi detaily.

V případě, že uživatel najede myší na agregovanou čáru, zobrazí se mu v popisu čáry jen jméno uzlu, na kterém jsou služby měřeny, protože čára reprezentuje několik služeb zároveň a není tedy možné zobrazovat podrobnosti o jedné konkrétní službě. Zobrazení jména uzlu je užitečné v případě, že má uživatel do vizualizace přidáno více sekundárních uzlů. Pomocí popisu pak pozná, služby kterého uzlu agregovaná čára reprezentuje.

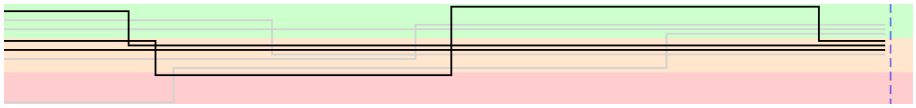
## 6.2 Heart-beat mód vs. explorační mód

Vizualizace nabízí dva způsoby průzkumu dat. **Heart-beat mód** vizualizace je situace, kdy je časový bod ukotven v přítomnosti (u pravého okraje grafu) a celý graf postupně *„ubíhá“* doleva. Při jakémkoliv pohybu časového bodu mimo přítomnost (ať už tahem přímo časového bodu nebo posunutím celého grafu) se vizualizace přepne do **exploračního módu**. V tomto módu se graf vizualizace samovolně nehýbe a uživatel si sám volí, kterou část grafu chce sledovat.

Heart-beat mód je možné obnovit přesunutím časového bodu zpět do přítomnosti nebo kliknutím na tlačítko Heart-beat.

## 6.3 Přidávání sekundárních uzlů

Do vizualizace lze zahrnout libovolný počet sekundárních uzlů (obr. 6.11). Jejich služby jsou pak vizualizovány šedými čarami (zatímco služby primárního uzlu jsou vizualizovány černými čarami).



Obrázek 6.11: Graf po přidání sekundárního uzlu

Přidání sekundárních uzlů je užitečné ve chvíli, kdy chce uživatel srovnat jevy na více uzlech.

### 6.3.1 Přidání uzlů vybraných uživatelem

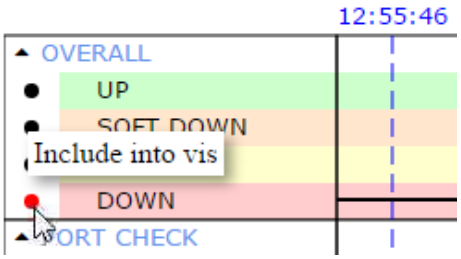
Pokud chce uživatel zahrnout do vizualizace konkrétní sekundární uzly, pak si může snadno vybrat, které to budou. V mém mockupu síťové topologie se to dá udělat držením klávesy ctrl a klikem na vybraný uzel. Po propojení s reálnou vizualizací síťové topologie může být tato funkce naprogramována jinak.

### 6.3.2 Přidání všech uzlů se službou ve vybraném stavu

Existuje ještě druhý způsob, jak přidat do vizualizace sekundární uzly. Při tomto způsobu si ale uživatel sám nevybírá konkrétní uzly, které chce přidat, ale přidá všechny uzly vyhovující jisté podmínce.

Tou podmínkou je, že v čase určeném časovým bodem má uzel alespoň jednu službu z vybrané kategorie ve vybraném stavu. Tyto uzly uživatel přidá do vizualizace kliknutím na kruh, který se nachází vedle názvu vybraného stavu ve vybrané kategorii (obr. 6.12).

Tímto může uživatel snadno zjistit, na kterých všech uzlech se daný jev vyskytl ve stejném čase.




Obrázek 6.12: Zahrnutí sekundárních uzlů majících alespoň jednu službu z kategorie OVERALL v čase 12:55:46 ve stavu DOWN

## 6.4 Konfigurační dialog

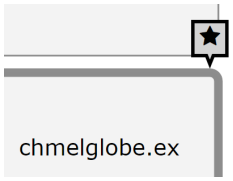
Konfigurační dialog nabízí nastavení upozornění a nastavení barevného pozadí pod stavy v grafu. Je dostupný přes tlačítko Config, ale po integraci vizualizace do projektu bude dostupný přes rozhraní KYPO, do kterého bude vizualizace zasazena.

### 6.4.1 Nastavení upozornění

Uživatel si může nastavit, že chce být upozorňován na to, že se služba, kterou sleduje, ocitla v určitém stavu. Upozornění je zobrazováno ve vizualizaci topologie sítě, takže je uživatel upozorněn i ve chvíli, kdy nemá zobrazenou vizualizaci jevů na uzlu, na kterém se služba měří.

Existují dvě formy upozornění. Upozornění **bublinou** (Bubble notification) je zobrazováno jako komiksová bublina nad uzlem (obr. 6.13). Je primárně zamýšleno jako krátkodobé upozornění, které po chvíli zmizí. Upozorňuje tedy primárně na přechod služby do daného stavu. Bublina obsahuje ikonku, kterou si uživatel vybírá při konfiguraci ze seznamu.

Čas od přechodu sledované služby do daného stavu, po kterém má bublina zmizet, se také nastavuje v konfiguračním dialogu a dá se nastavit i možnost, že bubliny nebudou mizet po určitém čase, ale až služba opustí daný stav. Čas, po jehož uplynutí bubliny mizí, je pro všechna nastavená upozornění stejný.




Obrázek 6.13: Upozornění bublinou

Druhou formou upozornění (obr. 6.14) je upozornění pomocí **barevného zvýraznění rohu uzlu** (Corner notification). Toto upozornění zůstává zobrazeno dokud se sledovaná služba nachází v daném stavu. Je tedy primárně určeno pro přehled stavu klíčových služeb sítě. Barvu trojúhelníku si uživatel vybírá v konfiguračním dialogu.

Uživatel může toto upozornění skrýt klikem na něj.


Obrázek 6.14: Upozornění barevným trojúhelníkem v rohu uzlu

U obou typů upozornění lze také nastavit poznámku, jejíž text se zobrazí v tooltipu po najetí myší na upozornění.

Pokud se vizualizace nachází v exploračním módu, nezobrazují se upozornění odpovídající současnému okamžiku, ale upozornění odpovídající okamžiku určenému časový bodem.

Vzhled upozornění je v tuto chvíli pouze provizorní. Tyto upozornění budou součástí vizualizace síťové topologie, která v této práci není použita, a jejich vzhled bude tedy muset být naimplementován v rámci ní.

### 6.4.2 Nastavení barevného pozadí pod stavy

V konfiguračním dialogu je také možné nastavit barvy pozadí stavů v grafu (obr. 6.15).

Obrázek 6.15: Graf se změněnými barvami pozadí

### 6.4.3 Uložení a načtení nastavení

Vizualizace se spustí ve výchozím nastavení. Uživatel má ale možnost si své nastavení uložit do souboru a poté jej znovu načíst. Nastavení se ukládá ve formátu JSON[2].

2. `http://www.json.org/`

# 7 Použité technologie

K vytvoření vizualizace jsou použity webové technologie. Konkrétně jde o jazyky HTML, SVG, CSS a JavaScript, který je momentálně zřejmě nejrozšířenějším nástrojem pro programování interaktivních funkcí na webu [10]. Veliká část implementace je prováděna pomocí JavaScriptové knihovny **D3.js**, která nabízí nástroje značně zjednodušující tvorbu interaktivních vizualizací.

## 7.1 JavaScript

JavaScript je multiplatformí objektově orientovaný skriptovací jazyk. Je většinou využívaný pro programování interaktivních prvků na internetových stránkách. Příkladem využití JavaScriptu mimo internetové stránky jsou například rozšíření programu Adobe Acrobat. [11]

Jedná se o interpretovaný jazyk, ale na rozdíl od jiných interpretovaných jazyků se jeho program spouští až po stažení internetové stránky na straně klienta (a ne tedy na straně serveru). Z toho plynou jistá bezpečnostní omezení [12], protože práce programu na straně klienta může ohrozit uživatelova data. JavaScript je možné spustit i na straně serveru, ale není to tak obvyklé. [13]

Jazyk byl poprvé standardizován společností ECMA v roce 1997. Od té doby vyšlo několik nových standardů – poslední v roce 2015. [14]

## 7.2 D3

D3[1] (zkratka pro Data-Driven Documents) je knihovna programovacího jazyka JavaScript. D3 nabízí funkce usnadňující transformaci dat na prvky značkovacího souboru [15]. V konkrétní rovině můžeme mluvit o tom, že pomocí knihovny D3 můžeme snadno z dat vytvářet vizualizaci v jazyce SVG, která se pak typicky stává součástí HTML dokumentu.

Vyvinul ji Mike Bostock s kolektivem v roce 2011. Jedním z členů týmu, který D3 vyvinul, byl i Jeffrey Heer, který před prací na D3 už vyvinul několik nástrojů pro vizualizaci dat na webu – prefuse,

1. `https://d3js.org/`

Flare a Protovis (poslední jmenovaný vyvíjel již společně s Mikem Bostockem). [3]

Knihovna D3 je vydána pod licencí BSD, která umožňuje volné šíření licencovaného obsahu, přičemž vyžaduje pouze uvedení autora a informace o licenci, spolu s upozorněním na zřeknutí se odpovědnosti za dílo. [16]

## 7.3 Font Awesome

Font Awesome[2] je sada škálovatelných vektorových ikon založená na technologiích CSS a Less. [17] Jedná se o druhý nejrozšířenější nástroj pro zobrazování ikon na webu, hned po sadě Google Font API společnosti Google. [18]

Font Awesome vytvořil Dave Gandy a je vydán pod licencí SIL OFL 1.1, která umožňuje jeho volné šíření. [19]

V této práci je sada Font Awesome použita pro všechny ikony, které jsou ve vizualizaci použity.

## 7.4 File Saver

FileSaver.js[3] je knihovna jazyka JavaScript poskytující funkce pro snadné ukládání souborů na straně klienta [20].

Knihovnu File Saver vytvořil Eli Grey a publikoval ji pod licencí MIT, která umožňuje její volné šíření, pod podmínkou uvedení autora a licence. [21]

V této práci jsou funkce knihovny File Saver použity k ukládání nastavení konfigurace do souboru.

## 7.5 HTML, SVG a CSS

Volba těchto technologií byla přirozená, protože použití knihovny D3.js a jazyka JavaScript je s nimi úzce propojeno.

2. `http://fontawesome.io/`
3. `https://github.com/eligrey/FileSaver.js`

HTML je značkovací jazyk používaný především pro tvorbu webových stránek. V této práci je použit pro obalení vizualizace, aby mohla být zobrazena internetovým prohlížečem.

SVG je značkovací jazyk používaný pro popis dvojrozměrné vektorové grafiky. Pomocí jazyka SVG byla vytvářena grafická část vizualizace.

CSS je jazyk používaný pro popis způsobu zobrazení prvků jazyků HTML a SVG na webových stránkách. Pomocí jazyka CSS byly vizualizaci přiřazeny základní vlastnosti týkající se grafického zobrazení.

# 8 Implementace

V této kapitole jsou shrnuty klíčové části implementace. Především tvorba generátoru dat a některé problémy řešené při implementaci vizualizace jevů a jejich stavových změn.

## 8.1 Generátor dat

Jak už bylo řečeno v předchozích kapitolách, v průběhu této práce ještě nebylo vyřešeno dodávání dat z reálných měření. Pro účely této práce bylo tedy potřeba vytvořit simulaci těchto měření, neboli generátor dat. Ten bude při integraci vizualizace do projektu KYPO nahrazen pravým zdrojem dat. Generátor byl vytvořen v jazyce JavaScript a jeho kód je umístěn v samostatném souboru[1].

Všechny funkce a proměnné týkající se generátoru jsem uložil do objektu "generator", abych předešel kolizím v názvech proměnných mezi různými soubory.

### 8.1.1 Popis práce generátoru

Generátor obsahuje napevno zakomponovaný seznam měřených služeb. Tento seznam jsem získal od týmu CSIRT-MU a služby na něm uvedené odpovídají těm, které se budou měřit ve skutečnosti. To proto, abychom již při práci se simulovanými daty pracovali s podobným množstvím a strukturou dat, jaké můžeme očekávat ve skutečnosti. To nám pomohlo odhalit některé nedostatky vizualizace již při vývoji a zvolit včas jiné řešení problému.

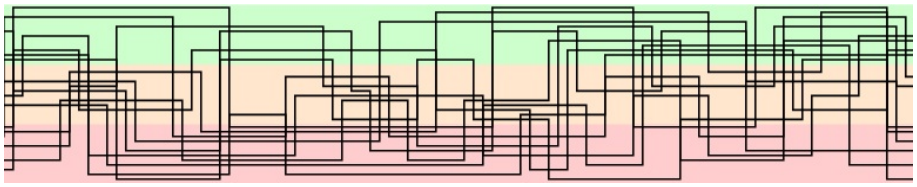
Generátor v pravidelném časovém intervalu vytváří pro každou ze služeb na seznamu **údaj o měření stavu služby**. Ten obsahuje dvě položky. Stav, ve kterém se služba právě nachází, a čas měření.

### 8.1.2 Výběr stavu

Stav služby byl původně vybírán náhodně. To ale znamenalo, že se stav každé služby měnil skoro při každém měření. Vizualizace byla pak velmi nepřehledná (obr. 8.1) a bylo potřeba definovat určitá pravidla,

1. src/dummy_data_generator.js

která by napomohla přiblížení chování simulace reálnému chování služeb.

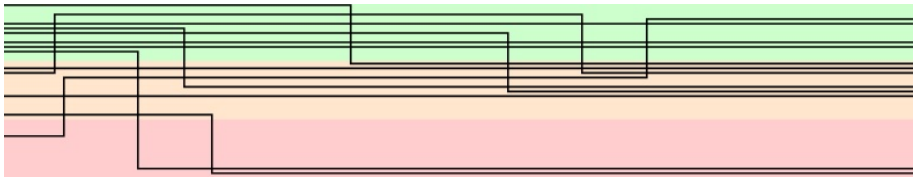


Obrázek 8.1: Vizualizace při změně stavu služby při každém měření

Při konzultaci s Tomášem Plesníkem mi bylo řečeno, že stav služeb se v realitě nebude měnit každých 10 vteřin (jak tomu bylo v případě takto naprogramovaného generátoru), ale spíše jednou za několik desítek minut. Takto jsem ale generátor nastavit nechtěl, protože by to bylo nevhodné pro rychlou odezvu a testování chování generátoru při různých situacích.

Nakonec jsem tedy do kódu zavedl proměnnou noChangeProbability, které lze nastavit hodnotu od 0 do 1. Tato proměnná určuje pravděpodobnost, se kterou se služba bude vyskytovat ve stejném stavu, jako při předchozím měření. Tímto jsem mohl při vývoji snadno ovlivnit, jak často se bude stav služeb měnit. Pokud nastane změna, nový stav se pak vybírá náhodně.

Fakt, že se stav služeb nebude měnit tak často, vyřešil problém s nepřehledností vizualizace (obr. 8.2).



Obrázek 8.2: Vizualizace při změně stavu služby s pravděpodobností 0,15

### 8.1.3 Výběr stavu služby overall

Většina služeb se může vyskytovat ve třech stavech — OK, UNKNOWN a CRITICAL. Výjimku tvoří služba **overall**, která se měří na každém uzlu a představuje souhrnný stav uzlu. Ve své podstatě tedy vlastně nejde o službu, ale pro jednoduchost to tak nazývejme.

Služba overall se může vyskytovat ve čtyřech stavech — UP, SOFT DOWN, SOFT UP a DOWN. Mezi těmito stavy ale nepřechází libovolně. Měření celkového stavu uzlu má výsledek buď kladný, anebo záporný. Při prvním měření přiřadíme službě buď stav UP (kladný výsledek), anebo DOWN (záporný výsledek). Pokud je služba ve stavu UP a výsledek následujícího měření je záporný, tak služba nepřejde přímo do stavu DOWN, ale nejprve do stavu SOFT DOWN. Do stavu DOWN se služba dostane až po třech záporných výsledcích v řadě. Pokud se mezitím objeví kladný výsledek měření, pak se služba okamžitě vrací ze stavu SOFT DOWN do stavu UP.

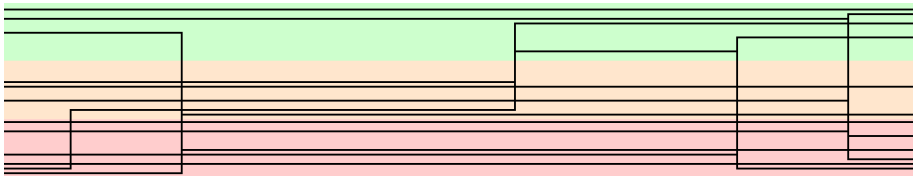
Obdobně pokud je stav služby DOWN, tak při kladném výsledku se stav změní na SOFT UP a až po třech kladných výsledcích v řadě se stav změní na UP.

Důvodem tohoto způsobu přiřazování stavů službě overall je to, že tato služba je klíčová pro vyhodnocování cvičení. Pokud by nastala chyba v měření nebo v přenosu informace o měření, mohl by se stav uzlu vyhodnotit nesprávně. To by mohlo výrazně ovlivnit vyhodnocování cvičení. Proto je zde tato pojistka, která výrazně snižuje pravděpodobnost chybného určení stavu (chyba v měření by musela nastat třikrát po sobě).

### 8.1.4 Čas měření

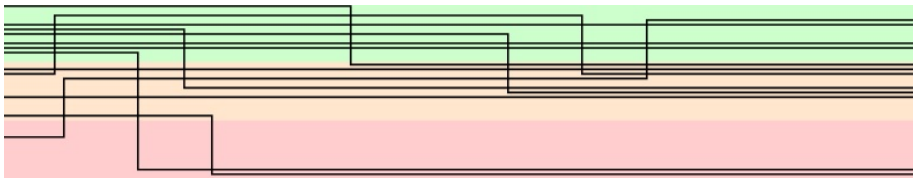
Čas měření bude v praxi samozřejmě odpovídat času, kdy bylo měření provedeno. Při generování falešných dat byl původně jako čas měření nastavován čas generování daného údaje. Údaj se ukládá s přesností na vteřiny a počítačové vygenerování nového údaje o měření probíhá tak rychle, že čas měření byl u všech služeb vždy stejný. Ve vizualizaci se objevil problém s tím, že všechny služby přecházely do jiného stavu ve stejný čas a čáry vykreslující jejich přechod se tedy překrývaly. To

bylo velmi nepřehledné, protože uživatel nebyl při překrytí schopný na první pohled určit, do jakého stavu služba přešla (obr. 8.3).



Obrázek 8.3: Vizualizace při měření všech služeb ve stejný čas

Na konzultaci mi ale bylo řečeno, že v praxi se stav všech služeb neměří ve stejný čas a můžu tedy generátor nastavit tak, aby čas měření služeb nastavoval různě. To celou vizualizaci velice zpřehlednilo (obr. 8.4).



Obrázek 8.4: Vizualizace při měření služeb v různé časy

### 8.1.5 Shrnutí

Generátor byl navržen tak, aby se jeho chování co nejvíce podobalo reálnému měření stavů služeb, s výjimkou situací, kdy by tato podobnost výrazně komplikovala práci na implementaci.

Hlavní rozdíly, ve kterých se chování generátoru odlišuje od reálného měření stavů služeb, jsou:

- Měření jsou prováděna mnohem častěji. V generátoru je to každých 10 vteřin, v praxi to bude přibližně jednou za 6 minut.
- Četnost služeb, které se nachází ve stavech OK, UNKNOWN a CRITICAL, je přibližně stejná. V praxi bude zřejmě většina služeb jedné kategorie ve stejném stavu.

- Služby nebudou v praxi měnit svůj stav tak často.
- V praxi bude overall stav uzlu nějakým způsobem korespondovat se stavy služeb daného uzlu. V generátoru je overall stav uzlu určován nezávisle na stavu služeb uzlu.

## 8.2 Vizualizace jevů a jejich stavových změn

Všechny HTML prvky vizualizace jevů a jejich stavových změn jsou obalené prvkem . Díky tomu se dá snadno celá vizualizace z HTML stránky odebrat, což se využívá např. při změně výběru sledovaného uzlu.

Kód vizualizace obsahuje hned v úvodu dva objekty obsahující veškeré **nastavení vizualizace**. Objekt config obsahuje obecná nastavení (šířka vizualizace, časový interval aktualizování grafu, atd.) a objekt userConfig obsahuje nastavení, které může uživatel během práce s vizualizací měnit (přiblížení, upozornění, skrývání kategorií, jazyk nápovědy, atd.).

Celá vizualizace je naimplementovaná ve třech souborech. Soubor single_node_vis.js obsahuje kód způsobilý za vykreslení a funkcionalitu samotné vizualizace, kód v souboru config_dialog.js zajišťuje vykreslení a funkcionalitu konfiguračního dialogu a soubor manual.js obsahuje kód vykreslující nápovědu.

### 8.2.1 Posouvání grafu

Při vizualizování grafu z naměřených hodnot jsem řešil otázku, jakým způsobem graf naimplementovat, aby při posouvání grafu nebylo nutné přepočítávat všechny souřadnice čar. Graf se totiž posouvá velice často (ať už samovolně při heart-beat módu vizualizace, anebo uživatelovým přičiněním) a časté počítání souřadnic čar by mohlo způsobit vysoké vytížení procesoru.

Moje implementace je tedy taková, že všechny linky reprezentující měřené služby[2] jsou obalené SVG prvkem <g> (dále jen skupina), jemuž je pomocí atributu transform nastavena pozice odpovídající nejdřívějšímu času, který se v naměřených datech vyskytuje. Tento

2. SVG prvek <polyline>

čas má pak relativně ke skupině vodorovnou souřadnici 0 a všechny ostatní časy mají vodorovné souřadnice vyšší.

Pomocí atributu clip-path je pak skupině nastavena viditelná oblast jen na oblast vizualizace (obr. 8.5), takže se zobrazuje jen část grafu mezi levým a pravým okrajem vizualizace.

Pokud je třeba posunout graf, pak stačí jen změnit atribut transform skupiny. Díky tomu, že souřadnice všech prvků obsažených ve skupině se určují relativně k transformaci skupiny, se pak posunou na obrazovce i všechny prvky ve skupině obsažené.

### 8.2.2 Skrývání kategorií a agregace stavů

Další problém, který jsem při implementaci řešil, byl jak naimplementovat SVG prvky tak, aby se při zobrazení/skrytí kategorie (nebo podkategorie) nebo při agregaci stavu nemusela vykreslovat znovu celá vizualizace a zároveň aby se nemusely nastavovat nové souřadnice pro všechny prvky vizualizace. Tento problém jsem vyřešil následujícím vrstvením HTML prvků:

```
<svg>
  <g id="cat0">
    <g id="cat0state0">
      <g id="cat0state1">
        ...
      </g>
    </g>
    <g id="cat1">
      ...
      <g id="cat2">
        ...
      </g>
    </g>
  </g>
</svg>
```

Skupina kategorie cat2 tedy není potomkem prvku <svg>, ale potomkem skupiny kategorie cat1 a ta je potomkem skupiny kategorie cat0.

Každá z těchto skupin obsahuje prvky související s danou kategorií nebo s daným stavem a každá z těchto skupin má také nastavenou transformaci pomocí atributu transform.

Pokud tedy chci například skrýt kategorii cat0, pak stačí udělat následující změny:

1. Nastavit skupinu cat0state0 jako neviditelnou.
2. Nastavit novou transformaci skupiny cat1.

Všechny stavy kategorie cat0 jsou součástí skupiny cat0state0. Díky tomu se zneviditelní úplně všechny stavy dané kategorie.

Všechny další kategorie (cat2, cat3, atd.) jsou součástí skupiny cat1, takže nastavením nové transformace skupiny cat1 posunu nejen kategorii cat1, ale i všechny následující kategorie, protože jejich souřadnice se zadávají relativně k rodičovskému prvku (tedy i skupině cat1).

Bez této struktury by program musel vykreslovat celou vizualizaci znovu, při každé takového změně, anebo by musel nastavovat novou transformaci u všech skupin a nejen skupiny následující. Takováto řešení by ale zbytečně zvýšila počet výpočtů prováděných procesorem.

Obdobný systém je použitý i u skrývání/zobrazování podkategorií a agregování stavů.

# 9 Uživatelské testování

Po dokončení všech funkčních prvků vizualizace jsem provedl uživatelské testování. Toho se zúčastnilo osm dobrovolníků. Protože cílovými uživateli aplikace nebudou běžní uživatelé, ale pouze zaměstnanci CSIRT-MU, kteří jsou do problematiky zasvěceni, bylo zbytečné do testování zahrnovat laiky. Tři z účastníků testování byli tedy přímo zaměstnanci CSIRT-MU, kteří budou s aplikací v praxi opravdu pracovat, a dalších pět uživatelů byli studenti nebo absolventi Fakulty informatiky Masarykovy univerzity nebo Fakulty informačních technologií Vysokého učení technického.

## 9.1 Průběh testování

V úvodu testu jsem uživatele uvedl do kontextu, ve kterém se bude vizualizace využívat. Když bylo testování prováděno se zaměstnancem CSIRT-MU, byla tato část velice krátká, protože se v tomto kontextu velice dobře vyzná. Když bylo testování prováděno s jiným dobrovolníkem, bylo třeba mu stručně vysvětlit, co projekt KYPO vlastně dělá, jak vypadá situace, kdy má zájem si tuto vizualizaci zobrazit, a co může přibližně očekávat, že v ní najde (tj. seznámit jej vůbec s významem měřených dat). Vycházel jsem přitom z toho, že v praxi s vizualizací nebude pracovat člověk, který netuší, co se v ní odehrává a těmto podmínkám jsem chtěl přiblížit i uživatelské testování.

Následně jsem dal uživateli 5 až 10 minut na to, aby se s vizualizací sám co nejvíce seznámil. Aby zkoušel přijít na různé funkce vizualizace, odhadoval, co tyto funkce přesně dělají, a celkově se tedy zkusil zorientovat v novém prostředí. Upozornil jsem jej také na to, že je u vizualizace k dispozici nápověda a řekl jsem mu, ať se ji nebojí využít.

Po uplynutí tohoto času jsem dal uživateli papír, na kterém bylo 13 úkolů, které se měl pokusit splnit. Tímto jsem si ověřoval pochopitelnost všech hlavních funkcí vizualizace.

Uživateli jsem předal také dotazník, kde po splnění (nebo nesplnění) každého z úkolu uvedl, jak obtížné pro něj bylo tento úkol splnit. Na výběr měl ze 4 možností:

- Snadné po seznámení se s aplikací
- Snadné po přečtení nápovědy
- Obtížné, ale nakonec jsem to zvládl/a.
- Obtížné. Na řešení jsem nepřišel/nepřišla.

Seznam všech 13 úkolů a vyhodnocení odpovědí najdete v příloze B.

Hlavní zpětnou vazbou pro mě bylo sledování jejich postupu a komentáře, připomínky a návrhy, které mi během testování sdělovali. Před začátkem testu jsem je totiž vyzval, ale mluvili co nejvíc o tom, co je během testování napadá a nad čím přemýšlejí. Tyto připomínky jsem si po celou dobu cvičení zaznamenával.

## 9.2 Výsledky testování

Testování ověřilo, že vizualizace se již nachází ve stavu, kdy je možné ji začít používat, nicméně bylo vzneseno také veliké množství připomínek a návrhů na přidání další funkcionality nebo na vylepšení té stávající.

Hlavním závěrem, který z testování vyvozuji je, že naprogramovaná funkcionalita nabízí hodně možností a je poměrně snadno pochopitelná, ovšem největším problémem vizualizace je špatná grafická úprava rozhraní (ne vizualizace samotné). Ta mohla výrazně znesnadnit orientaci v prostředí, a tedy i vypracování zadaných úkolů během testování.

Uživatelé, kteří nemají zkušenost s projektem KYPO, občas během testování nepochopili přesně zadání úkolu a snažili se tedy vypracovat něco jiného, než bylo cílem. Nicméně následné testy se zaměstnanci CSIRT-MU ukázaly, že tohle zmatení vycházelo jen z neznalosti kontextu a v praxi se jej tedy není potřeba obávat.

Některé z menších připomínek (např. popis tlačítek, nevhodné ikony, atd.) jsem po uživatelském testování už do implementace zapracoval. Návrhy na výraznější změny zde popisuji podrobněji.

### 9.2.1 Popis a filtrace služeb

Uživatelé při testu okamžitě pochopili, že po najetí myší na čáru se jim zobrazí popisek, identifikující službu, kterou čára reprezentuje. Nicméně ve chvíli, kdy měli za úkol najít konkrétní službu v kategorii, kde bylo služeb více, si stěžovali na to, že musí čáry prohlížet jednu po druhé a hledat službu, kterou chtějí najít, tímto primitivním způsobem.

Řešením by bylo dát uživateli možnost, zobrazit si na okraji grafu popisky všech vykreslovaných služeb. Kliknutím nebo najetím na tento popisek by se služba okamžitě zvýraznila.

S touto funkcionalitou se počítalo už od začátku práce, akorát zatím nebyla naimplementována. Kód vizualizace je pro ni nicméně připraven a to tak, že objekt `userConfig` obsahuje proměnné `labelLeft` a `labelRight`, které nabývají hodnot `true` a `false`. Pokud je hodnota proměnných nastavena na `true`, vykresluje se pak ve vizualizaci na okraji grafu prostor, pro tyto popisky služeb. Samotné vytvoření popisků a funkcionality s nimi spojené je však třeba teprve naimplementovat.

Tohle řešení mi připadá vhodné. Úskalí vidím v tom, že bude potřeba vhodně zvolit formátování popisků tak, aby byly i na malém prostoru dobře čitelné.

Dále uživatelé zmiňovali, že by se jim líbilo, kdyby byla možnost některé služby z grafu skrýt. Uživatel by se tedy mohl i v rámci jedné kategorie (podkategorie) zaměřit pouze na služby, které ho zajímají.

Tuto funkcionalitu bych při případné implementaci přidal právě k popiskům služeb. Služby by se z vizualizace mohly vyřadit např. tlačítkem u popisku nebo přetáhnutím popisku mimo oblast vizualizace.

### 9.2.2 Orientace v čase

Uživatelé měli často problém s orientací v čase v rámci grafu. Jediný časový údaj, co se momentálně v grafu nachází, je časový bod (modrá svislá přerušovaná čára). Mnozí uživatelé navíc ani nepřišli na to, že se tento bod dá posouvat.

Řešení, které nás napadlo už při implementaci, je přidat ke grafu časovou osu. Tohle řešení jsem zkoušel naimplementovat pomocí knihovny D3.js, která pro tvorbu os obsahuje nástroje. Osu jsem implementoval tak, že jsem jako její okraje zadal časy, odpovídající okrajům

zobrazované části grafu a knihovna D3.js už poté osu vytvořila automaticky. Problém ale byl, že jsem tyto časy zadával jen s přesností na vteřiny, přičemž jedna vteřina byla v grafu reprezentována často několika pixely. To způsobovalo to, že zatímco graf se pohyboval plynule, časová osa se pohybovala sekaně, jen jednou za několik pohybů grafu. Čas uvedený na ose byl sice správný, ale nesourodost v posouvání osy a grafu působila velice rušivě, proto jsem nakonec tuto implementaci opustil a ve vizualizaci tedy teď žádná časová osa není.

Tento problém by šel vyřešit několika způsoby, například:

- Zadáváním časů okrajů grafu do funkcí knihovny D3.js s větší přesností.
- Vytvořením takové osy (pomocí knihovny D3.js), která by popisovala celý rozsah grafu (tedy i jeho neviditelnou část) a která by se pohybovala společně s grafem. Stejně jako u grafu by byla viditelná část osy nastavena atributem `clip-path`.
- Vytvořením vlastní časové osy bez použití knihovny D3.js.

Uživatelé při testování také navrhli další způsoby, jak by se mohl v grafu zobrazovat čas. Jednou z možností by bylo zobrazovat čas v popisku služby. Tento čas by pak odpovídal pozici myši. Druhou možností, která by mohla orientaci v čase usnadnit, je, že by se po kliknutí myší do oblasti grafu na toto místo posunul časový bod. Nevýhodou tohoto řešení je podle mě to, že posun časového bodu způsobuje přerušení heart-beat módu vizualizace, což by bylo nežádoucí, když chce uživatel heart-beat mód při zjišťování času zachovat. Já osobně bych navrhl řešení, že by se do grafu mohl vykreslovat ještě sekundární časový bod, který by byl vykreslený nějakou méně výraznou barvou a pohyboval by se neustále do pozice, kde se právě nachází kurzor myši. Poté co kurzor opustí oblast grafu, by tento pomocný časový bod zmizel.

### 9.2.3 Přehled nad grafem

K přehledu nad grafem se při testování objevila jedna připomínka a dva návrhy nové funkcionality.

Připomínkou, kterou uvedli dva z osmi účastníků testování, bylo, že je pro ně matoucí fakt, že se v přehledu zmenšuje oranžová plocha

(reprezentující zobrazovaný časový úsek) a šedá plocha (reprezentující celkový čas měření) zůstává stejně veliká, zatímco v realitě zobrazovaný časový úsek zůstává stejně veliký a celkový čas měření se zvětšuje.

Byly vysloveny návrhy, implementovat přehled tak, aby oranžová plocha zůstávala stejně veliká a naopak šedá plocha se zvětšovala, což by odpovídalo tomu, co se děje reálně.

Tohle řešení mi ale nepřipadá vhodné, protože šedá plocha by po chvíli přetekla mimo oblast vizualizace. Alternativou by bylo, že by se šedá plocha po přetečení nezobrazovala celá, ale jen její úsek, který by bylo možné posouvat. To mi ale také nepřipadá jako dobré řešení, protože je pak znemožněno uživateli upravit oranžovou plochu tak, aby pokryla celý čas cvičení.

Vzhledem k tomu, že šesti z osmi účastníků testování tahle funkcionalita nepřipadala matoucí, bych se přikláněl k zachování současné implementace.

Při testování více uživatelů uvedlo, že by bylo dobré, aby se čas zobrazovaný grafem dal nastavit nejen tahem okraje oranžové plochy, ale také přímým zadáním okrajových časů. V této funkcionalitě vidím jen výhody a určitě ji doporučují k implementaci.

Dále jeden z účastníků uvedl, že by bylo dobré mít u přehledu tlačítka, pomocí kterých by mohl zobrazit posledních 15 minut, 30 minut, 1 hodinu, atd., cvičení. Tato funkce mi připadá užitečná, ale bylo by dobré tento nápad před implementací ještě probrat s lidmi z CSIRT-MU a zjistit, jak moc by takovou funkcionalitu využívali a jestli by tlačítka jen zbytečně nezaplňovala prostor.

### 9.2.4 Konfigurační dialog

Ovládání konfiguračního dialogu bylo všem uživatelům poměrně jasné.

Největším problémem je především to, že pokud chce uživatel dialog opustit bez uložení změn, dialog jej na to neupozorní. Tlačítko Save není na první pohled viditelné a změny tedy často nemusí být uloženy. V případě, že se uživatel chystá opustit konfigurační dialog bez uložení změn, by měla aplikace zobrazit upozornění typu "Opravdu chcete odejít bez uložení změn?". Uživatel by tedy byl upozorněn a mohl by se rozhodnout, zda chce změny uložit anebo zrušit.

Stejný problém nastává i v případě přidávání nového upozornění. Uživatelé si často nevšimli, že po vyplnění všech kolonek týkajících se nového upozornění musejí ještě kliknout na tlačítko Add, kterým nové upozornění přidají do seznamu těch již existujících. Tenhle problém navrhuji vyřešit tak, že by se řádek umožňující přidat nové upozornění zobrazil až po kliknutí na tlačítko typu Add new notification. Zároveň by se při přidávání nového upozornění skryla tlačítka umožňující opuštění dialogu.

Jeden z osmi účastníků cvičení uvedl, že mu existence dvou druhů upozornění (bubliny a rohy) připadá zbytečná. Uvedl, že pokud hlavní rozdíl mezi těmito dvěma druhy upozornění je ten, že bubliny po čase zmizí a rohy zůstávají zobrazeny, pak by mu ve chvíli, kdy je možné nastavit čas mizení bublin jako nekonečný (a vyrovnat tím tedy jejich význam významu rohů), připadalo vhodnější mít jeden způsob upozornění s tím, že by šel nastavit čas mizení pro každé upozornění zvlášť. Některá by tedy uživatel mohl nechat po čase zmizet a některá by mohl nechávat zobrazena.

Zda jsou opravdu potřeba dva typy upozornění nebo jen jeden se ukáže zřejmě až v budoucnu, při používání vizualizace v praxi.

### 9.2.5 Sekundární uzly

Ohledně práce se sekundárními uzly vznikly při testování tři připomínky.

První z nich byla, že při přidání více sekundárních uzlů se všechny jejich služby vykreslují šedou barvou, nezávisle na tom, ke kterému z přidaných uzlů služba patří. Jediná možnost jak to zjistit, je tedy pomocí popisku služby. Návrh byl, že by se službám sekundárních uzlů mohly přiřazovat různé barvy. Návrh mi v zásadě připadá rozumný, ale bylo by potřeba vyřešit to, aby byly takto obarvené čáry na pozadí grafu zřetelně vidět. Muselo by se brát v potaz i to, že uživatel má možnost nastavit si barvu pozadí pod jednotlivými stavy. Uzel může mít služby mnoha různých kategorií, tyto kategorie můžou mít mnoho různých stavů a barvy pod všemi těmito stavy si uživatel může nastavit dle svého uvážení. Najít tolik barev, které by v případě této extrémní situace byly jasně viditelné, by mohlo být velice složité. Řešením by mohlo být to, že si uživatel tyto barví zvolí sám. Tato volba by

mohla být realizována např. pomocí tlačítek umístěných nad oblastí grafu.

Další připomínkou bylo, že by bylo vhodné, aby měl uživatel možnost zachovat množinu uzlů, jejichž služby se vizualizují a pouze změnit výběr primárního uzlu. To v momentální implementaci není možné, protože ve chvíli, kdy uživatel vybere nový primární uzel, se automaticky vyprázdní množina sekundárních uzlů. V současné implementaci je primární uzel volen klikem levého tlačítka myši. Při použití reálné vizualizace síťové topologie bych navrhl vybírat primární uzel pomocí kontextového menu dostupného pravým tlačítkem myši. V tomto kontextovém menu by mohla být i možnost změnit sekundární uzel na uzel primární se zachováním množiny vykreslovaných uzlů.

Poslední připomínkou k práci se sekundárními uzly byla ta, že funkcionalita kolečka vedle jména stavu (přidání uzlů splňujících danou podmínku) je velice neintuitivní. Bylo by vhodné použít jinou ikonu než kolečko nebo případně celou funkci naimplementovat přehledněji. (Např. klikem na kolečko by se uzly okamžitě nepřidaly, ale otevřel by se dialog, kde by si uživatel mohl nastavit podmínku, kterou musí přidávaný uzel splňovat.)

### 9.2.6 Nápověda

Nápověda byla pro uživatele při testování vítaným pomocníkem, nicméně bylo by vhodné ji pro větší přehlednost lépe strukturovat a přidat do ní obrázky.

Také by bylo vhodné přidat více nápovědy i k prvkům samotné vizualizace, pomocí tzv. tooltipů, aby uživatel při každé nejasnosti nemusel hledat radu v dlouhém manuálu.

### 9.2.7 Přehled počtu služeb v jednotlivých stavech

Při testování vznikl také návrh, že by se v záhlaví každé kategorie mohla vypisovat informace o počtu služeb v jednotlivých stavech této kategorie. Tento nápad mi v zásadě nepřipadá špatný, ale muselo by se vyřešit, jestli by se vypisovaly i služby sekundárních uzlů, případně jakým způsobem.

### 9.2.8 Přenositelnost mezi prohlížeči

Vizualizace byla vyvíjena za použití internetového prohlížeče Google Chrome 58.0.3029.110. Při uživatelském testování bylo zjištěno, že některé naimplementované funkce nejsou podporovány jinými druhy prohlížečů.

Například při otevření vizualizace v prohlížeči Mozilla Firefox 51.0.1 nefungovalo přibližování a oddalování grafu a v konfiguračním dialogu byly špatně vykreslovány ikony z fontu Font Awesome.

# 10 Integrace do projektu KYPO

Vizualizace byla vyvíjena mimo prostředí projektu KYPO. V této kapitole je shrnuto, co všechno se musí udělat pro integraci vizualizace do projektu.

Všechny vizualizace jsou do KYPO zahrnuty formou portletu. Protože v rámci portálu KYPO nejde o tzv. standalone aplikaci, bude potřeba *„obalit"* portletem i tuto vizualizaci.

Dále bude při integraci potřeba změnit některé části kódu vizualizace. V souboru single_node_vis.js je hned v úvodu definován objekt config, který obsahuje proměnné, které ovlivňují základní nastavení vizualizace. Tyto proměnné je možné změnit, pokud bude potřeba upravit základní vlastnosti vizualizace. Jednu z těchto proměnných bude potřeba změnit určitě a to konkrétně updateTime. Tato proměnná definuje časový interval, po kterém se aktualizuje celý graf. Pro potřeby implementace byl tento čas nastaven na 5 vteřin, ale v praxi se bude měření dostupnosti služeb provádět pravděpodobně jen jednou za 6 minut a není tedy důvod pro to, aby se graf aktualizoval tak často.

Hned za objektem config se nachází objekt userConfig, který obsahuje proměnné, které může uživatel během chodu aplikace měnit. V tomto objektu bych při integraci doporučil změnit výchozí hodnotu proměnné secPerPx. Ta definuje počet vteřin, které jsou reprezentovány jedním pixelem (tedy přiblížení grafu). Současně nastavená hodnota 0.1 se může ukázat jako zbytečně malá ve chvíli, kdy aktualizace grafu neprobíhá tak často.

V kódu souboru single_node_vis.js je také jedenáct funkcí, které budou při integraci do KYPO vyžadovat pozornost. Dvě z nich zajišťují dodávání dat do vizualizace a devět z nich obsluhuje vykreslování upozornění ve vizualizaci síťové topologie. V této práci se při dodávání dat i při vizualizaci síťové topologie pracuje pouze s mockupy. Proto bude potřeba funkce přepsat ve chvíli, kdy se mockupy nahradí reálným zdrojem dat a reálnou vizualizací topologie sítě.

Všechny tyto funkce jsou umístěny na začátku souboru a jsou označeny komentářem REWRITE FUNCTION, díky čemuž jsou snadno vyhledatelné fulltextovým vyhledáváním.

## 10.1 Vstupní data

V současné době jsou data do vizualizace dodávána z generátoru dat, který pouze simuluje měření stavů služeb. V realitě budou dodávána jiným způsobem (z databáze).

Při programování vizualizace bylo myšleno na to, že se zdroj dat změní, a jediné, co je třeba přeprogramovat, jsou funkce getData(node) a funkce getAllData(), které zajišťují dodávání dat do vizualizace. Funkce getData vrací pole dat týkajících se služeb měřených na zadaném uzlu a funkce getAllData vrací pole úplně všech naměřených dat.

### 10.1.1 Formát dat

Dodávaná data musí splňovat předepsaný formát. Jeden datový záznam může vypadat například takto:

```
{
  "service" : "RPC",
  "category" : "PORT CHECK",
  "subcategory" : null,
  "dst_hostname" : "srv-dc",
  "src_hostname" : "desk-user1",
  "port" : "135",
  "protocol" : "TCP",
  "history" : [
    {
      "state" : "OK",
      "timestamp" : "2016-09-22-11-30-00"
    },
    {
      "state" : "CRITICAL",
      "timestamp" : "2016-09-22-11-30-12"
    }
  ]
}
```

- service – název měřené služby

- category – název kategorie měřené služby
- subcategory – název podkategorie měřené služby (nepovinná položka)
- dst_hostname – název uzlu, na kterém je služba měřena
- src_hostname – název uzlu, ze kterého je služba dotazována
- port – port, přes který je služba dotazována
- protocol – protokol, pomocí kterého je služba dotazována
- history – historie měření stavu služby
  - state – stav služby
  - timestamp – čas, kdy byla služba měřena, ve formátu yyyy-MM-dd-hh-mm-ss

V tomto formátu musí být data nejpozději při výstupu z funkcí getData a getAllData. V databázi se data ukládají v mírně odlišném formátu, takže bude potřeba je přeparsovat.

## 10.2 Provázání s vizualizací síťové topologie

Některé funkce vizualizace jevů a jejich stavových změn a vizualizace síťové topologie spolu souvisejí. Při práci s mockupem síťové topologie tedy nebylo možné tyto funkce naprogramovat přesně tak, jak budou muset být naprogramovány při práci s pravou vizualizací.

Některé z těchto funkcí jsou obsluhovány vizualizací síťové topologie a budou tedy muset být do vizualizace naimplementovány. Jiné vycházejí z vizualizace jevů, ale jejich činnost ovlivňuje grafické zobrazení vizualizace síťové topologie. Tyto funkce již v kódu vizualizace jevů existují, ale budou muset být přepsány. Všechny tyto funkce jsou velice jednoduché a jejich přeprogramování by tedy nemělo být problémové.

### 10.2.1 Zobrazení vizualizace jevů a jejich stavových změn

Nejzákladnější funkcionalitou, která musí být naimplementována do vizualizace topologie sítě, je, aby si uživatel v topologii mohl vybrat uzel, pro který chce vizualizaci jevů zobrazit. Dále je také třeba, aby měl možnost přidat do vizualizace pomocí topologie i sekundární uzly.

Pro zobrazení vizualizace se používá funkce showVis(primary, secondary), která vytvoří celou vizualizaci jevů pro zadaný primární uzel a pole sekundárních uzlů. Tato funkce vytvoří úplně novou vizualizaci, takže pokud byla předtím již nějaká vytvořena, je třeba ji z webové stránky odstranit (např. při změně výběru primárního uzlu). Případně, pokud by mělo být na webové stránce zobrazeno více vizualizací jevů zároveň, je potřeba oddělit jmenné prostory jednotlivých vizualizací.

### 10.2.2 Zvýrazňování uzlů v topologii

V kódu vizualizace jevů je několik funkcí způsobujících různá zvýraznění uzlů v topologii. Tyto funkce bude také potřeba přeprogramovat ve chvíli, kdy bude mockup topologie nahrazen reálnou vizualizací.

Jde konkrétně o tyto funkce:

- highlightNode(node) – zvýrazní vybraný uzel v síťové topologii.
- unHighlightNode(node) – zruší zvýraznění uzlu v síťové topologii.
- unHighlightAllNodes() – zruší zvýraznění všech uzlů v síťové topologii.
- markAsSecondary(node) – označí vybraný uzel v síťové topologii jako sekundární uzel.
- unmarkAsSecondary(node) – vybraný uzel již nebude v síťové topologii označen jako sekundární uzel.

### 10.2.3 Zobrazování upozornění v topologii

Dále kód vizualizace jevů obsahuje funkce zajišťující vykreslení a smazání nastavených upozornění ve vizualizaci síťové topologie. I tyto funkce bude třeba přeprogramovat.

Jde o funkce:

- showBubble(rule) – vykreslí komiksovou bublinu u uzlu, podle zadaného pravidla. Platnost pravidla se již ověřovat nemusí – funkce je volána pouze ve chvíli, kdy už je pravidlo ověřeno.
- deleteBubbles() – smaže z vizualizace topologie všechna vykreslená upozornění komiksovou bublinou.
- showCorner(rule) – zajišťuje vykreslení barevného trojúhelníku v rohu uzlu, podle zadaného pravidla. Platnost pravidla se již ověřovat nemusí – funkce je volána pouze ve chvíli, kdy už je pravidlo ověřeno.
- deleteCorners() – smaže z vizualizace topologie všechna vykreslená upozornění barevným trojúhelníkem v rohu uzlu.

## 10.3 Vstupní konfigurovatelné soubory

Implementace obsahuje soubory categories.js a bubbleIcons.js. Tyto soubory v sobě mají uloženou tu část kódu, která se bude muset měnit při aplikaci na nový způsob využití vizualizace a bude se tedy měnit častěji, než zbytek kódu. Proto byl tento kód umístěn do zvláštních souborů – aby byl snáze dohledatelný a konfigurovatelný.

Každý z těchto souborů obsahuje jen definici jedné proměnné, která je pak ve zbytku kódu používána.

### 10.3.1 categories.js

Soubor categories.js obsahuje definici proměnné allCategories. Jedná se o pole objektů, kde každý objekt obsahuje proměnné category a states (obr. 10.1).

V tomto poli musí být uvedeny všechny kategorie, které se ve vizualizovaných datech mohou vyskytnout. U každé kategorie pak

musí být uveden výčet stavů, ve kterých se služby z dané kategorie mohou nacházet.

Pokud budou v souboru uvedeny i kategorie, které se v datech vyskytovat vůbec nebudou, není to pro aplikaci problém. Je tedy možné do souboru jen přidávat kategorie nehledě na to, pro jaký okruh dat je vizualizace aktuálně využívána.

Podkategorie se do výčtu neuvádějí, protože je předpokládáno, že výčet možných stavů u podkategorií je stejný, jako u jejich rodičovských kategorií.

Potřebu, mít předem daný výčet stavů pro každou kategorii, bych rád demonstroval následujícím příkladem. Cvičení (a tedy i měření dostupností služeb) začalo v 11:00. Uživatel chce v 11:20 zobrazit vizualizaci jevů a jejich stavových směn. Aplikace získá z databáze data o službách vybraného uzlu. Celkový stav uzlu (kategorie OVERALL) byl ale celých 20 minut pouze UP. Aplikace tedy nemá žádné informace o tom, jaké jiné stavy by se mohly v kategorii OVERALL vyskytnout a nezbývá jí, než vykreslit ve vizualizaci pouze jeden možný stav u kategorie OVERALL. Když se ale v čase 11:21 celkový stav uzlu dostane do stavu SOFT DOWN, aplikace by buď neměla tento stav kam vykreslit, anebo by musela přidat do vizualizace nový řádek. To by za prvé působilo nesourodě, protože by vizualizace během chodu měnila samovolně vzhled svého rozhraní, a za druhé by to předem neposkytlo uživateli informaci o tom, které všechny stavy můžou u dané služby nastat. Proto je důležité, aby aplikace znala předem výčet všech možných stavů dané kategorie.

Tento výčet není vhodné napevno zakomponovat přímo do kódu aplikace, protože se při aplikaci na jiný typ dat mohou objevit nové kategorie a je tedy důležité mít možnost tyto kategorie do výčtu snadno přidat. Proto jsem se rozhodl výčet umístit do samostatného, přehledného souboru.

```
var allCategories = [
  {
    "category" : "OVERALL",
    "states" : ["UP", "SOFT DOWN",
                "SOFT UP", "DOWN"]
  },
  {
    "category" : "PORT CHECK",
    "states" : ["OK", "UNKNOWN", "CRITICAL"]
  }
];
```

Obrázek 10.1: Příklad obsahu souboru categories.js

### 10.3.2 bubbleIcons.js

Soubor bubbleIcons.js obsahuje pouze definici proměnné fontAwesomeIcons. Jde o pole obsahující výčet ikon, které jsou uživateli nabízeny při nastavování upozornění bublinou (obr. 10.2).

Ikony se vybírají z fontu Font Awesome, v jehož systému má každá ikona svůj název. Pro potřeby této aplikace je ale potřeba znát unicode hodnotu této ikony a ne její název. Tyto hodnoty se pak ukládají do pole v souboru bubbleIcons.js.

```
fontAwesomeIcons = [
  "\uf004",
  "\uf005",
  "\uf006",
  "\uf007",
  "\uf008",
  "\uf009"
];
```

Obrázek 10.2: Příklad obsahu souboru bubbleIcons.js

# 11 Závěr

Tato práce se zabývala vytvořením vizualizace jevů a jejich stavových změn pro projekt KYPO podle předem vytvořeného návrhu. Byl požadavek na to, aby vytvořená vizualizace byla snadno přizpůsobitelná novému druhu vstupních dat. Toho bylo docíleno obecnou implementací, nezaměřující se na jeden konkrétní případ využití, a přítomností vstupních konfigurovatelných souborů.

Vizualizace byla vyvíjena nad simulovanými daty a propojena byla pouze s mockupem vizualizace síťové topologie. Bude tedy potřeba ještě zajistit dodávání dat a provázání obou vizualizací (síťové topologie a této). Poté je možné vizualizaci začít používat, ale prospěly by jí ještě menší změny a vylepšení. Největší současnou slabinou je vizuální stránka ovládacích prvků. Vizualizace samotná je přehledná.

Vizualizace jevů a jejich stavových změn slibuje projektu KYPO výrazné zjednodušení dohledu nad počítačovou sítí. Největší uplatnění najde pravděpodobně při cvičeních Cyber Czech, kdy ji budou používat rozhodčí udělující jednotlivým týmům body.

Na závěr bych si dovolil navrhnout několik rozšíření stávající funkcionality.

## 11.1 Možnost rozšíření - statický mód

Současná implementace počítá pouze s tím, že vizualizace bude zobrazována v reálném čase, během průběhu měření služeb a pravý okraj grafu tedy vždy odpovídá současnosti. Je třeba ale myslet i na případ, kdy bude chtít uživatel zobrazit cvičení, které už skončilo.

Při aplikaci dat z takového cvičení na současnou implementaci bychom sice vizualizaci viděli, ale na pravém okraji grafu bychom měli velikou a nadbytečnou část grafu. Současná implementace totiž pracuje tak, že pokud bylo poslední měření stavu služby například v čase 09:00 a aktuálně je 17:00, pak vizualizace počítá s tím, že se služba nachází po celých 8 hodin ve stejném stavu, jako byla při posledním měření. Což není nic nekorektního, ale jestli měření stavů v 9:10 skončilo, je velice otravné, že v pravé části grafu je 7 hodin a 50 minut dat, která jsou naprosto zbytečná, a tato oblast se stále zvětšuje, protože graf je při každé aktualizaci vykreslován až po aktuální čas.

Vzhledem k tomu, že se bude vizualizace používat i k zobrazení záznamů z již proběhlých cvičení, bylo by vhodné, aby měl uživatel možnost spustit vizualizaci v tzv. statickém módu. To znamená, že by měl možnost nastavit čas, ve kterém měření stavů skončilo. V tomto čase by pak končil i samotný graf.

### 11.1.1 Návrh řešení

V současné implementaci se často pracuje s aktuálním časem. V kódu je na těchto místech většinou uvedena proměnná $\mathrm{now}$, která obsahuje údaj o aktuálním čase.

Mým návrhem je tuto proměnnou nahradit funkcí $\mathrm{getEndTime}()$, která by, v případě že by vizualizace běžela v běžném módu, vracela aktuální čas, ale v případě že by vizualizace byla spuštěna ve statickém módu, by tato funkce vracela čas posledního měření zadaný uživatelem při spuštění vizualizace.

Tímto by se při zadání cílového času omezilo vykreslení grafu časem, který uživatel zadal.

V současné implementaci je taky po zobrazení vizualizace spuštěno pravidelné opakování série funkcí, které zajišťují aktualizaci dat a grafu. Tyhle funkce nebude při statickém módu potřeba spouštět opakovaně, ale bude stačit jedno spuštění.

## 11.2 Možnost rozšíření - upozornění součástí vizualizace síťové topologie

Upozornění se momentálně nastavují v konfiguračním dialogu vizualizace jevů a jsou kontrolována vždy k času, který je určen časovým bodem. Důvodem tohoto řešení je, že ve chvíli kdy je časový bod posouván do minulosti, má uživatel možnost sledovat, jaká upozornění se objevovala v čase, který právě prozkoumává.

Není ale myšleno na okamžik, kdy není žádná vizualizace jevů zobrazena a uživatel sleduje pouze topologii sítě. V takovou chvíli neexistuje žádný časový bod, podle kterého by se mohla upozornění kontrolovat.

Vzhledem k charakteru upozornění, by bylo vhodné umístit dialog umožňující jejich nastavení do vizualizace síťové topologie. Ve chvíli,

kdy není zobrazena žádná vizualizace jevů, by se pak upozornění kontrolovala vždy pro aktuální čas.

Uživatel by díky tomu měl přehled o službách, které chce sledovat, už v rámci vizualizace síťové topologie a vizualizaci jevů by mohl zobrazit až ve chvíli, kdy by byl upozorněn na jev, který ho zajímá podrobněji.

### 11.2.1 Návrh řešení

Funkce checkRules, která zajišťuje kontolu a následné vykreslení upozornění, aktuálně není volána s žádným parametrem. Navrhuji zavést parametr time, který by označoval čas, pro který se upozornění kontrolují.

Pokud by byla zobrazena vizualizace jevů, pak by kontrola pravidel byla prováděna stejně jako doposud (tj. při každé změně popisku časového bodu). Pokud by ale žádná vizualizace jevů zobrazena nebyla, pak by se kontrola pravidel spouštěla opakovaně každou vteřinu a pravidla by se kontrolovala v závislosti na aktuálním čase.

## 11.3 Možnost rozšíření - data buffer

Funkce getData a getAllData jsou volány poměrně dost často. Jsou spouštěny při každé aktualizaci grafu, při kontrole upozornění (často každou vteřinu) i při najetí myší na kolečko vedle jména stavu (zahrnutí sekundárních uzlů splňujících danou podmínku). Při použití generátoru dat toho nepředstavuje problém, ale ve chvíli, kdy se budou data získávat z databáze, by takhle časté dotazování bylo velice nevhodné z důvodu velikého vytížení databáze.

### 11.3.1 Návrh řešení

Bylo by vhodné vytvořit tzv. buffer dat. Šlo by o proměnnou typu pole, do které by se v pravidelných intervalech ukládala všechna data z databáze. Funkce getData a getAllData by pak získávaly data z tohoto bufferu a ne ze samotné databáze, čímž by se počet přístupu do databáze výrazně snížil.

# Bibliografie


[1] Katy Börner a David E. Polley. *Visual Insights*. Massachusetts Institute of Technology, 2014.

[2] Tomáš Marek. „Efektivní vizualizace dat se zaměřením na základní typy grafů“. Dipl. Masarykova Univerzita, 2014.

[3] Scott Murray. *Interactive Data Visualization for the Web*. O'Reilly Media, Inc., 2013.

[4] Ústav výpočetní techniky Masarykovy univerzity. *KYPO - Kybernetický polygon*. URL: https://www.kypo.cz/ (cit. 11. 05. 2017).

[5] Ústav výpočetní techniky Masarykovy univerzity. *Možnosti využití | KYPO - Kybernetický polygon*. URL: https://www.kypo.cz/use-cases (cit. 11. 05. 2017).

[6] Masarykova univerzita. *Cyber Czech | CSIRT-MU*. URL: https://csirt.muni.cz/projects/cyber-czech (cit. 22. 05. 2017).

[7] Tomáš Bobek. „Visualization of Node-related Events in Computer Networks“. Dipl. Masarykova Univerzita, 2015.

[8] Miroslava Jarešová. „Event Based Vis“. Mockup vizualizace jevů a jejich stavových změn.

[9] W. Aigner et al. *Visualization of Time-Oriented Data*. Springer-Verlag London Limited, 2011.

[10] Dan Rowinski. *It's Official: JavaScript Is The Most Commonly Used Programming Language On Earth*. 22. břez. 2016. URL: https://arc.applause.com/2016/03/22/javascript-is-the-worlds-dominant-programming-language/ (cit. 23. 05. 2017).

[11] Adobe Systems Incorporated. *JavaScript for Acrobat | Adobe Developer Connection*. 2012. URL: http://www.adobe.com/devnet/acrobat/javascript.html (cit. 11. 05. 2017).

[12] Stephen Chapman. *What Javascript Can Not Do*. 14. červ. 2014. URL: https://www.thoughtco.com/what-javascript-cannot-do-2037666 (cit. 22. 05. 2017).

[13] Davey Waterson. *Back to the Server: Server-Side JavaScript On The Rise*. 12. ún. 2016. URL: https://developer.mozilla.org/en-US/docs/Archive/Web/Server-Side_JavaScript/Walkthrough (cit. 22. 05. 2017).

[14] Nicolás Bevacqua. *The JavaScript Standard*. 2017. URL: https://ponyfoo.com/articles/standard (cit. 11. 05. 2017).

[15] Michael Bostock, Vadim Ogievetsky a Jeffrey Heer. „D3 Data-Driven Documents“. In: *IEEE Transactions on Visualization and Computer Graphics* 17 (pros. 2011), s. 2301–2309.

[16] Mike Bostock. *D3.js - Data-Driven Documents*. 2015. URL: https://d3js.org/ (cit. 11. 05. 2017).

[17] Dave Gandy. *Font Awesome, the iconic font and CSS toolkit*. URL: http://fontawesome.io/ (cit. 17. 05. 2017).

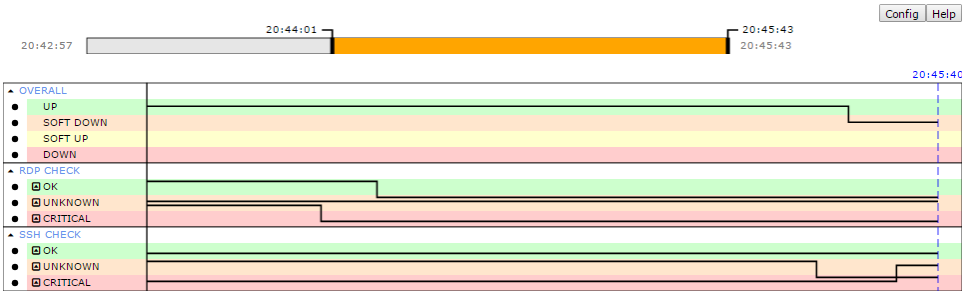
[18] Wappalyzer. *Wappalyzer - Font Scripts*. URL: https://wappalyzer.com/categories/font-scripts (cit. 17. 05. 2017).

[19] Dave Gandy. *Font Awesome License*. URL: http://fontawesome.io/license/ (cit. 17. 05. 2017).

[20] John Duprey. *Client Side File Saving with FileSaver.js*. 2015. URL: http://codepen.io/jduprey/details/xbale (cit. 23. 05. 2017).

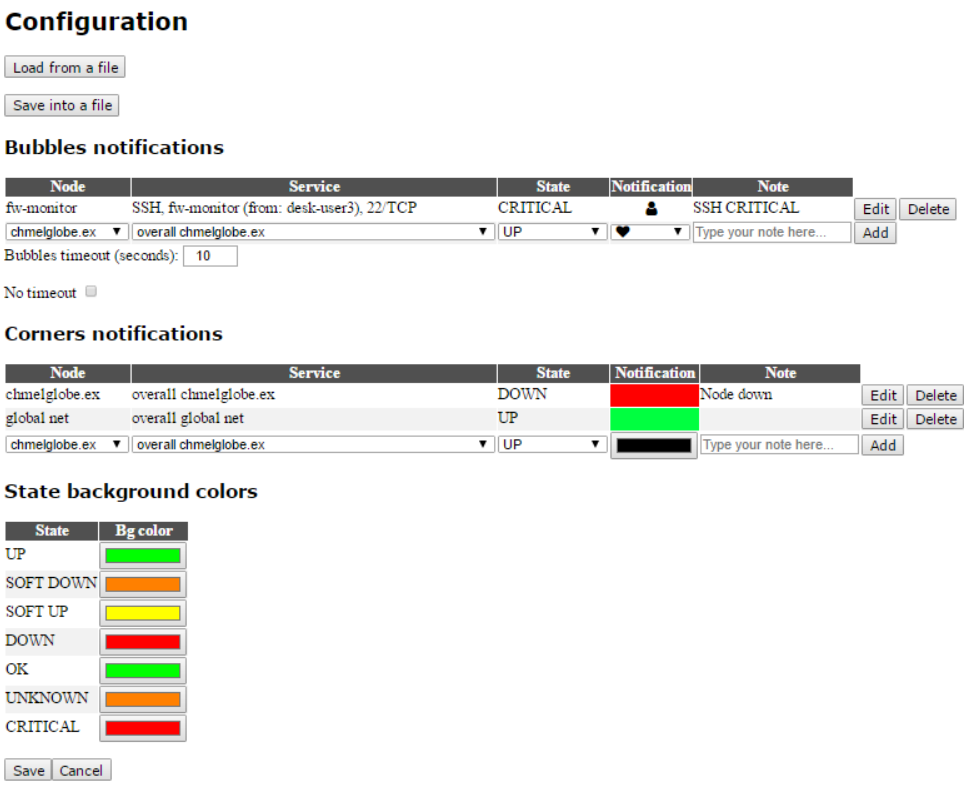
[21] Eli Grey. *FileSaver.js/LICENSE*. 2. zář. 2016. URL: https://github.com/eligrey/FileSaver.js/blob/master/LICENSE.md (cit. 11. 05. 2017).

# A Screenshoty




Obrázek A.1: Vizualizace jevů a jejich stavových změn


Obrázek A.2: Mockup vizualizace síťové topologie




Obrázek A.3: Konfigurační dialog

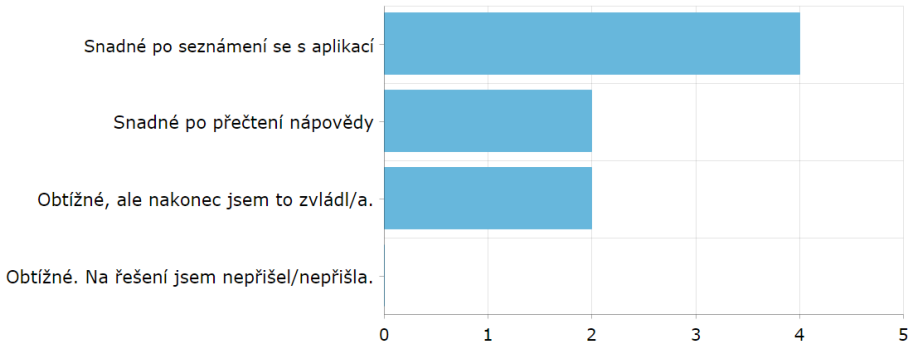
# B Úkoly zadané během uživatelského testování

Během uživatelského testování měli uživatelé za úkol pokusit se splnit 13 úkolů. Zde je uveden seznam těchto úkolů společně s vyhodnocením jejich obtížnosti na základě dotazníků vyplněných uživateli během testování.

1. **Zadání:** Zjistěte, v jakém čase byl celkový stav (tj. overall uzlu ICS poprvé ve stavu SOFT UP.

   **Důvod:** Tímto úkolem jsem si ověřoval, zda uživatelé pochopí funkcionalitu časového bodu a objeví možnosti, jak s ním manipulovat.

   **Výsledky:**

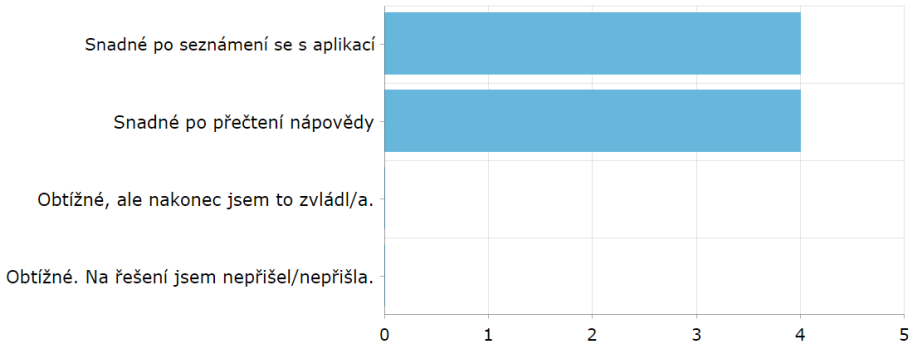



   Tento úkol byl pro některé uživatele obtížný především proto, že šlo o první úkol, kdy ještě nebyli s aplikací dostatečně seznámení. Při dalších úkolech už s manipulací časovým bodem neměli problém.

2. **Zadání:** Zjistěte, který z uzlů desk-user1, desk-user2 a desk-user3 se ocitl ve stavu SOFT DOWN (služba overall) nejdřív, a v jakém to bylo čase.

**Důvod:** Tímto úkolem jsem si ověřoval, zda uživatele napadne, že existuje možnost srovnat více uzlů v rámci jednoho grafu (přidání sekundárních uzlů).

**Výsledky:**

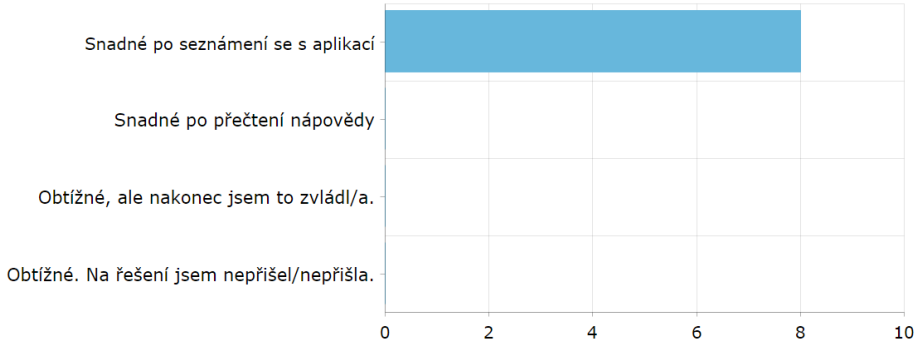



Uživatele většinou napadlo, že by mohli všechny tři uzly porovnat v rámci jednoho grafu a jak to udělat následně našli v nápovědě.

3. **Zadání:** Skryjte graf kategorie OVERALL.

**Důvod:** Tímto úkolem jsem si ověřoval, zda je pro uživatele intuitivní skrývání pomocí ikonky vedle názvu kategorie.

**Výsledky:**

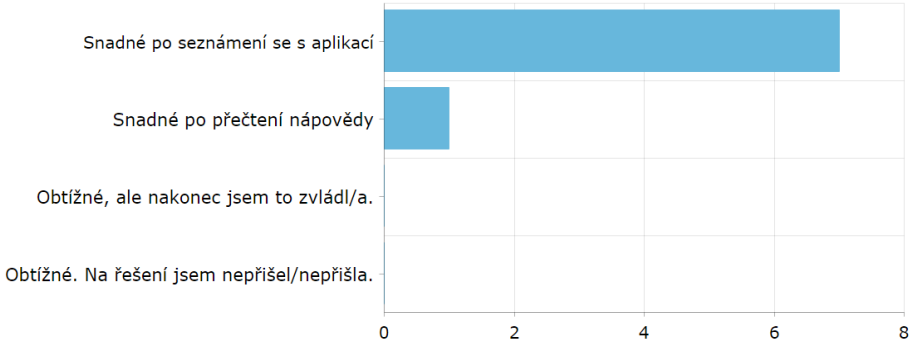



Tato funkcionalita nedělala žádnému z uživatelů problém.

4. **Zadání:** Nastavte pozadí pod stavem UNKNOWN na žlutou barvu.

   **Důvod:** Tímto úkolem jsem si ověřoval, zda uživatel najde funkcionalitu nastavení barvy pozadí v konfiguračním dialogu.

   **Výsledky:**



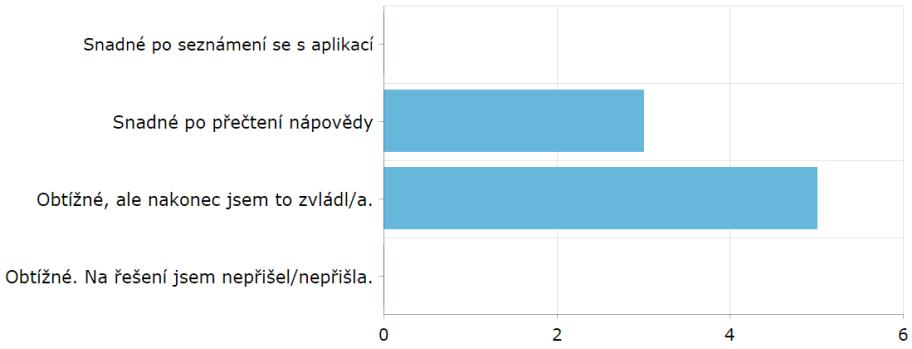

Někteří uživatelé první hledali možnost nastavení barvy pozadí klikem pravým tlačítkem myši na samotné pozadí grafu, ale

jinak bylo nalezení této funkcionality v konfiguračním dialogu pro uživatele snadné.

5. **Zadání:** Zobrazte vizualizaci služeb na uzlu DMZ a zahrňte do vizualizace (jako sekundární uzly) všechny uzly, jejichž celkový stav (tj. overall) byl v čase –:–:– DOWN.

   **Důvod:** Vzhledem k tomu, že jde o zřejmě nejméně intuitivní funkcionalitu celé vizualizace, zjišťoval jsem tímto úkolem převážně to, jestli bude pro uživatele snadné najít řešení pomocí nápovědy.

   **Výsledky:**

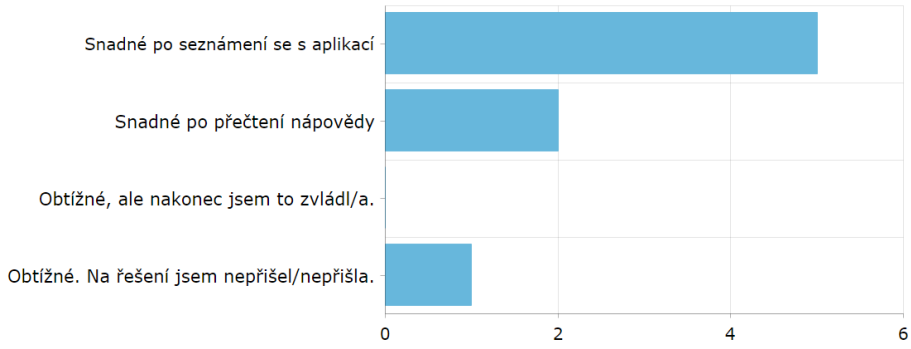



   U vypracování tohoto úkolu hrálo největší roli, jestli se uživatel snažil na řešení přijít sám nebo jestli byl zvyklý hledat pomoc v nápovědě.

6. **Zadání:** Nastavte vizualizaci tak, aby vás upozornila na to, že služba SSH uzlu fw-monitor dotazovaná z uzlu desk-user3 přes port 22 a protokol TCP přešla do stavu CRITICAL. Upozornění bude bublinou s ikonkou černé hvězdy a bude zobrazeno po dobu 20 vteřin.

   **Důvod:** Tímto úkolem jsem si ověřoval intuitivnost použití konfiguračního dialogu.

**Výsledky:**

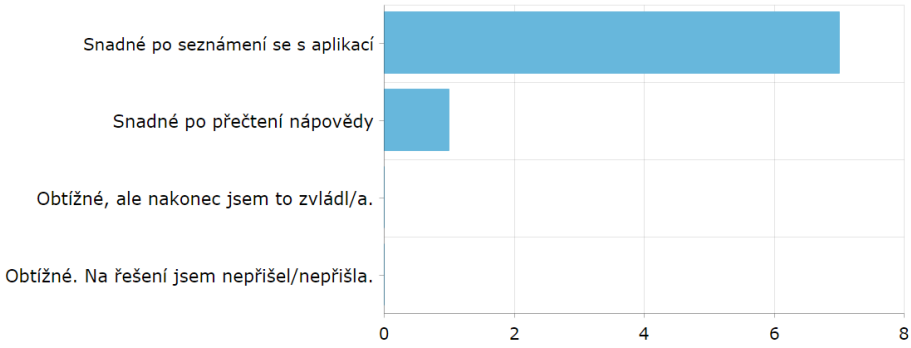



Většina uživatelů si všimla možnosti nastavení upozornění už při úvodním seznamování se s aplikací a splnění tohoto úkolu pro ně poté nebyl problém.

7. **Zadání:** Obnovte heart-beat mód vizualizace (tj. mód, kdy je časový okamžik ukotven v přítomnosti a vizualizace postupně „ubíhá“ doleva).

   **Důvod:** Tímto úkolem jsem si ověřoval intuitivnost tlačítka Heart-beat, případně posunutí časového bodu do přítomnosti.

   **Výsledky:**

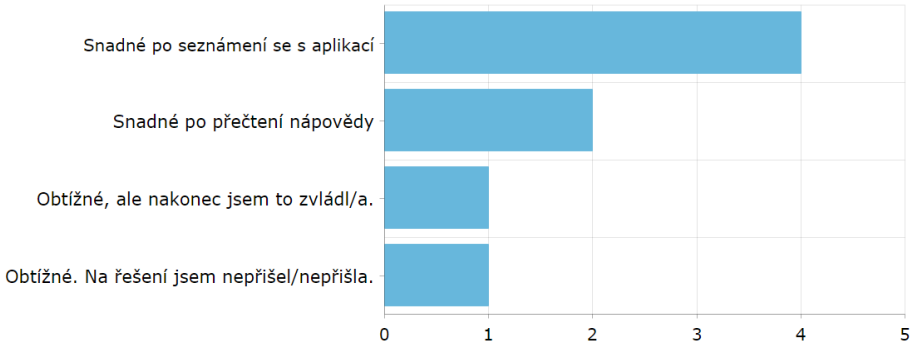


Splnění tohoto úkolu nepředstavovalo pro uživatele problém. Většinou si okamžitě všimli tlačítka Heart-beat.

8. **Zadání:** Bez počítání služeb odhadněte, v jakém stavu (OK, UNKNOWN, CRITICAL) se nachází nejvíce služeb uzlu srv-dc kategorie PRT CHECK podkategorie Kerberos v aktuálním čase.

   **Důvod:** Tímto úkolem jsem si ověřoval, zda uživatel pochopí význam agregace čar a napadne jej ji využít pro představu o množství služeb v jednotlivých stavech.

   **Výsledky:**

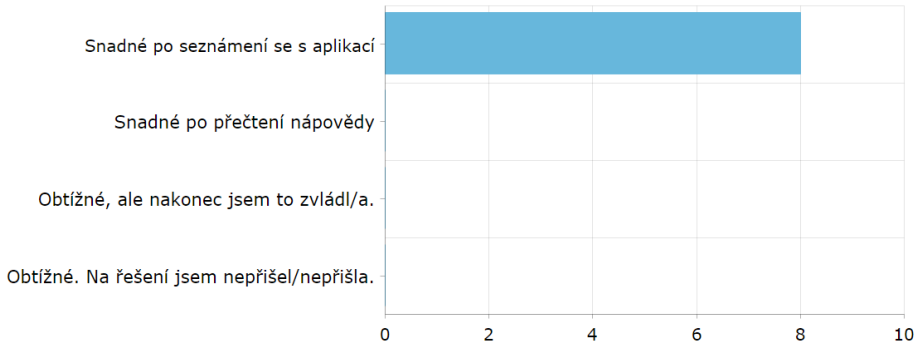



   U tohoto úkolu záleželo na tom, jestli si uživatel všiml agregace při úvodním seznamování se s vizualizací. Pokud ano, pak ji při tomto úkolu využil. Všichni uživatelé okamžitě pochopili, že tloušťka agregované čáry odpovídá počtu služeb, které se v daném stavu aktuálně nachází.

9. **Zadání:** Zobrazte pouze časový úsek od –:–:– do –:–:–.

   **Důvod:** Tímto úkolem jsem si ověřoval, zda uživatel pochopí význam přehledu nad grafem a zda je manipulace s ním intuitivní.

**Výsledky:**

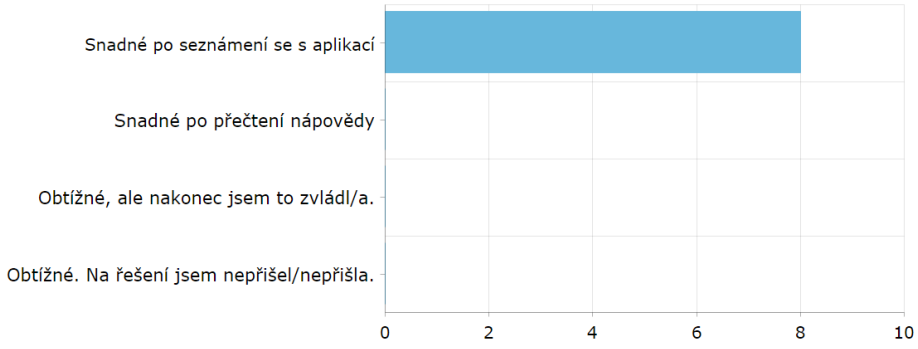



Řešení tohoto úkolu bylo pro uživatele snadné.

10. **Zadání:** Nastavte vizualizaci tak, aby vás upozornila na to, že služba RDP uzlu ics-historian dotazovaná z uzlu ics-workstation přes port 3389 a protokol TCP se vyskytuje ve stavu OK. Upozornění bude modrým trojúhelníkem v rohu uzlu ve vizualizaci topologie sítě a bude u něj uvedena poznámka „RDP OK“.

    **Důvod:** Tímto úkolem jsem si ověřoval intuitivnost použití konfiguračního dialogu.

    **Výsledky:**

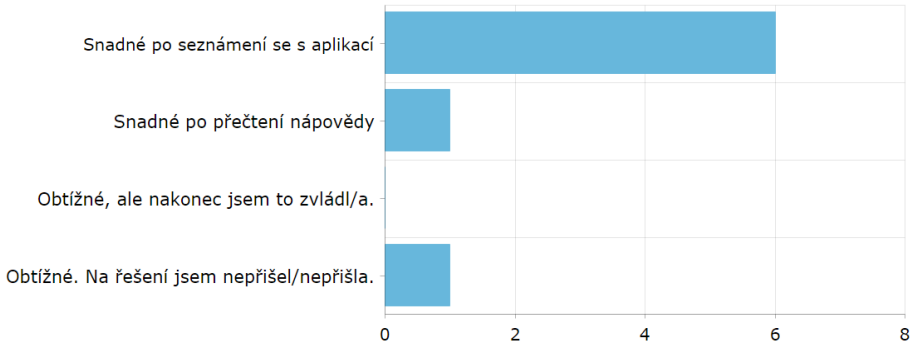


Řešení tohoto úkolu bylo pro uživatele snadné.

11. **Zadání:** Zjistěte, v jakém stavu byla služba ping (kategorie PING CHECK) uzlu DMZ, dotazovaná z uzlu global net přes port 80 a protokol ICMP v čase –:–:–.

    **Důvod:** Tímto úkolem jsem si ověřoval, zda bude pro uživatele jednoduché najít jednu konkrétní službu v kategorii, která obsahuje služeb několik.

    **Výsledky:**

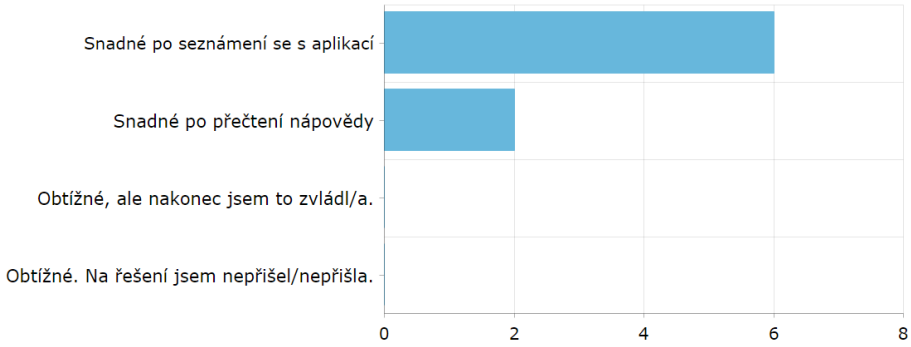



    Tento úkol dělal problém především uživatelům, kteří nejsou zaměstnanci CSIRT-MU, a to z důvodu neznalosti kontextu a následného nepochopení popisku u služby.

12. **Zadání:** Uložte do souboru aktuální nastavení.

    **Důvod:** Tímto úkolem jsem si ověřoval, jestli je tlačítko Save into a file v konfiguračním dialogu jasně viditelné a jeho funkce zřejmá.

**Výsledky:**

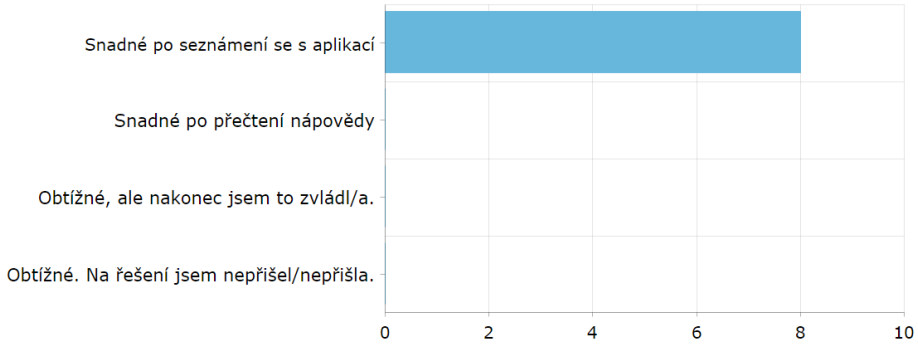



Tento úkol byl pro uživatele snadný.

13. **Zadání:** Aktualizujte webovou stránku a načtěte nastavení, které jste v předchozím bodě uložil/a.

    **Důvod:** Tímto úkolem jsem si ověřoval, jestli je tlačítko Load from a file v konfiguračním dialogu jasně viditelné a jeho funkce zřejmá.

    **Výsledky:**




Tento úkol byl pro uživatele snadný.

# C Elektronické přílohy

V elektronickém archivu této práce se nachází soubor priloha.rar obsahující zdrojový kód aplikace.

Vizualizace se zobrazí po otevření souboru index.html v internetovém prohlížeči. Složka css obsahuje CSS soubory definující kaskádové styly aplikace. Složka src obsahuje soubory se zdrojovými kódy v jazyce JavaScript.

Dále je součástí archivu soubor specifikace.pdf. Jde o prezentaci, která obsahovala původní návrh řešení, podle kterého jsem vizualizaci vytvářel.