Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Ilkovičova 3, 842 16 Bratislava 4

Manažment VoIP relácií

Dokumentácia riešenia projektu

Študijný program: Počítačové a komunikačné systémy a siete

Tím č.3: Bc. Jozef Baláž, Bc. Tomáš Boros, Bc. Adam Močkoř, Bc. Martin Pivarník, Bc. Matej Rybár, Bc. Timotej Tkáč

Vedúci tímového projektu: Ing. Ján Murányi

Ak. rok: 2013/14

Obsah

Zoznam obrázkov

1 Úvod

Predkladaný dokument obsahuje dokumentáciu k projektu Manažment VoIP relácií. Tento projekt sa rieši v rámci predmetu Tímový projekt.

V úvode dokumentu je uvedená motivácia pre riešenie manažmentu VoIP, podobné existujúce riešenia a zhrnutie výsledkov bakalárskej práce, z ktorej sa pri riešení vychádza.

Dokument ďalej obsahuje podrobnú analýzu jednotlivých funkčných prvkov a spôsobu komunikácie medzi nimi. V tejto časti je tiež opísaný spôsob využitia SIRUP-u pri riešení problému a analýza možností testovania výsledného systému v sieti.

Ďalšou časťou dokumentu je špecifikácia riešenia. Táto časť opisuje požiadavky na výsledný systém.

Opisuje jeho celkovú funkcionalitu , úlohu jednotlivých prvkov systému a tok dát medzi nimi.

Časť návrhu riešenia podrobnejšie špecifikuje funkcionalitu prvkov systému. Konkrétne sú opísané SIRUP zariadenia, ich rozmiestnenie, spôsob komunikácie a správy, ktoré sa pri komunikácií posielajú. V návrhu riešenia je tiež opísaný nástroj, ktorý umožní testovanie systému.

2 Zadanie

Multimediálne relácie založené na protokole SIP (Session Initiation Protocol) sa väčšinou skladajú z troch rôznych komunikačných kanálov (SIP, RTP a RTCP), pričom každý z týchto kanálov vyžaduje komunikáciu na osobitnom porte.

Táto skutočnosť sa odzrkadľuje na zložitom manažmente VoIP (Voice over IP) relácií.

Analyzujte architektúru „SIP Single Port“ a porovnajte ju s inými existujúcimi architektúrami zameranými na manažment VoIP relácií. SIP Single Port architektúra pre multimediálnu komunikáciu využíva na rozdiel od protokolu SIP len jeden port, čo je hlavnou motiváciou pre návrh optimalizácie manažmentu VoIP relácií.

Navrhnite aplikáciu, ktorá umožní riadiť aktívny manažment relácií. Výsledná aplikácia musí byť schopná adaptovať sa na náhle zmeny v sieti ako napríklad zahltenie, zhoršenie kvality, pád linky, zmena IP adresy klienta a podobne.

Na základe analýzy a návrhu implementujte aplikáciu a výsledok práce zhodnoťte.

3 Analýza

3.1 Manažment

Multimediálne relácie sa stali súčasťou nášho každodenného života a ich možnosti a kvalita pomáhajú medziľudskej komunikácii prekonávať veľké vzdialenosti s cieľom poskytnúť komunikujúcim stranám zážitok v najväčšej možnej miere podobný „živému“ kontaktu. Je žiaduce, aby zážitok z takejto komunikácie bol vždy na vysokej úrovni bez ohľadu na počet zúčastnených strán.

Na umožnenie účasti čo najvyššiemu počtu potenciálnych účastníkov v multimediálnej relácii je vhodné používať existujúce technológie so širokou dostupnosťou a podporou v koncových zariadeniach. Vhodným kandidátom je SIP protokol, ktorý je považovaný za štandard v danej oblasti.

Manažment multimediálnych relácií založených na SIP protokole pozostáva okrem iného aj z manažmentu kvality multimediálnych relácií, pričom pod týmto pojmom chápeme zabezpečenie potrebných zdrojov a parametrov v časti sieťovej infraštruktúry, ktorou prechádza dátový tok multimediálnej relácie. Ďalšou úlohou je správa sieťových filtrov (firewall), ktoré by v záujme bezpečnosti mali prepúšťať len požadovanú sieťovú komunikáciu a všetky ostatné komunikácie zahadzovať. Manažment multimediálnych relácií pokrýva aj profilovanie poskytovaných služieb, ktoré na základe identifikácie príjemcu poskytovanej služby sú poskytované v dohodnutom rozsahu a kvalite.

Vyššie popísané činnosti je jednoduché vykonávať nad reláciami medzi koncovými bodmi reprezentovanými jedinečnými IP adresami. Keďže množstvo IP adries je obmedzené, bolo nutné zaviesť koncept verejných a privátnych IP adries a preklad medzi nimi. Tento krok síce oddialil potrebu rozšírenia 32-bitového IP adresného priestoru, ale protihodnotou bolo znefunkčnenie niektorých sieťových protokolov vyšších vrstiev referenčného modelu ISO OSI a potreba zavedenia opravných mechanizmov, ktoré svoju úlohu plnia s čiastočným úspechom alebo naopak komunikáciu koncových bodov skomplikujú. Problémy nastávajú z dôvodu prekladu sieťových portov bez toho, aby na tento fakt boli komunikujúce koncové body upozornené.

Zabezpečenie kvalitatívnych parametrov prenosu dátového toku multimediálnej relácie je možné uskutočňovať na základe informácií protokolov nižších vrstiev referenčného modelu ISO OSI, ale tento prístup bez použitia výkonovo-náročnej hĺbkovej analýzy paketov nemusí vždy citlivo reagovať na požiadavky multimediálnej relácie a prípadné zlyhanie má za následok zlyhanie aplikácií využívajúcich čiastočne alebo úplne nefunkčnú časť siete. Preto je vhodné, aby aplikácia prevádzkujúca samotnú

multimediálnu reláciu mala prehľad o stave siete a vedela sa prispôsobiť prípadným výpadkom spojenia.

3.2 Kontext

Produkt tohto tímového projektu má za cieľ zjednodušiť manažment multimediálnych relácií založených na protokole SIP zjednodušením správy sieťových filtrov vyplývajúcej z použitia architektúry SIRUP popísanej v ďalších častiach dokumentácie. Rozšírením tejto architektúry o sledovanie vybraných metrík jednotlivých ciest v sieti a umožnením presunu multimediálnych relácií na inú dostupnú sieťovú cestu chceme predviesť schopnosť adaptovania aplikácie na náhle zmeny v sieti.

Aplikácia bude vhodná pre používateľov multimediálnych relácií založených na protokole SIP, konkrétne organizácie s redundantnou konektivitou do siete Internet, prípadne poskytovateľov internetového pripojenia a IP telefónie s redundantnou infraštruktúrou.

3.3 Existujúce riešenia

Relevantným príkladom existujúceho riešenia manažmentu multimediálnych relácií založených na SIP protokole je modul programu Kamailio s názvom Mediaproxy, ktorý umožňuje prechod multimediálnych relácií cez smerovače s prekladom sieťových adries a obmedzené profilovanie prichádzajúcich multimediálnych dátových tokov prostredníctvom manipulácie INVITE správ.

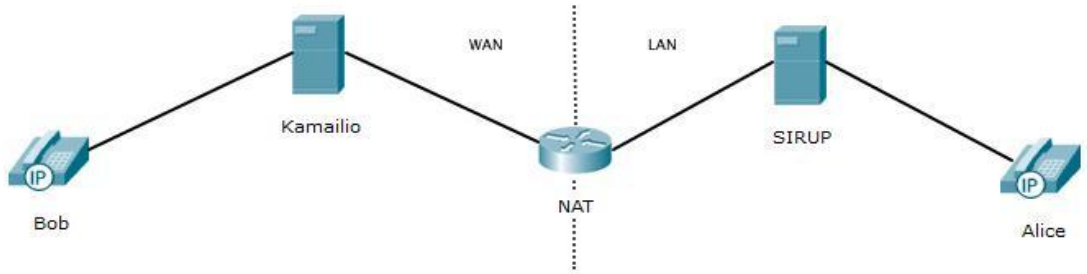
3.3.1 SIP Single Port

Téma tímového projektu vychádza z pôvodnej práce SIP Single Port. Účelom SIP Single Portu bolo umožniť prechod RTP a RTCP paketom v sieti s implementovaným prekladačom adries NAT. Nakoľko protokoly SIP, RTP a RTCP fungujú na troch rôznych portoch, prekladač adries ich videl ako tri rôzne komunikácie, a teda snahou bolo vytvoriť určitý preposielač, ktorý by tieto tri porty spojil do jedného a vytvoril tak z pohľadu prekladača jednu spoločnú komunikáciu. Riešenie bolo implementované a je funkčné.

3.3.2 Single a Dual Proxy režim

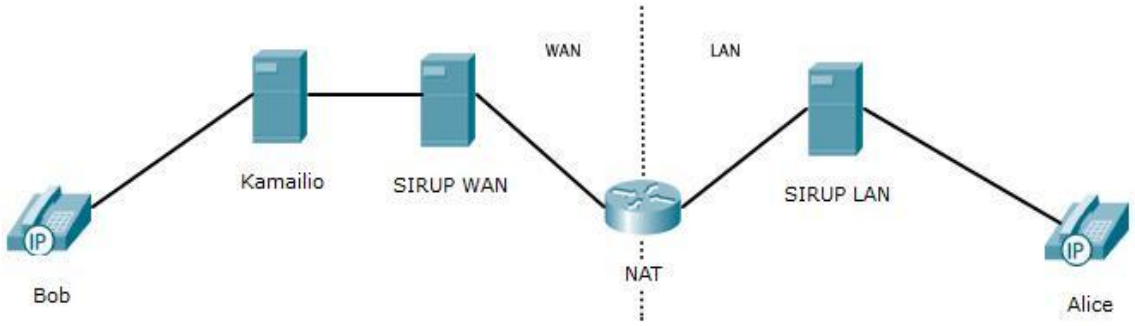
V závislosti od typu prekladu adries (preklad nezávislý od koncovej stanice, závislý od koncovej adresy, alebo závislý od koncovej adresy a portu) je používaný SIRUP (proxy server multiplexujúci pakety na jeden port) v režime single alebo dual proxy.

V režime single proxy je inštancia SIRUP-u len jedna a je umiestnená na strane klienta (režim prekladu nezávislého od koncovej stanice).



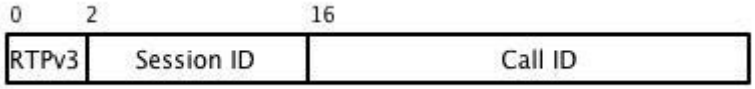
Obrázok 3-1 Single proxy režim

V prípade dual proxy režimu sú inštancie SIRUP-u dve (režimy prekladu závislého od koncovej adresy alebo závislého od koncovej adresy a portu), pričom jedna z nich je umiestnená v LAN sieti za prekladačom u klienta a druhá sa nachádza na druhej strane prekladača u poskytovateľa. Na oboch stranách sa pakety po prijatí multiplexujú na jeden port a následne sa posielajú na druhú stranu, kde sú znova demultiplexované a preposielané ďalej.



Obrázok 3-2 Dual proxy režim

Keďže sa viacero komunikácií zbieha do jedného portu medzi dvoma SIRUP-mi, nie je možné na druhej strane RTP a RTCP pakety spätne priradiť do jednotlivých komunikácií na základe portu ako zvyčajne, a teda je potrebné rámce značkovať. Prvé dva bity značky sú nastavené na jednotky tak, aby paket vyzeral ako RTPv3. Za týmito dvoma bitmi sa nachádza sessionID, ktorý určuje číslo relácie v rámci hovoru. Koniec značky tvorí callID. Táto značka sa umiestňuje hneď za UDP hlavičku.



Obrázok 3-3 SIRUP značkovanie

3.3.3 Funkcionalita SIRUP-u

Na to aby sme mohli realizovať manažment relácií potrebujeme dva navzájom kompatibilné proxy servery, a preto budeme od teraz uvažovať len o SIRUP-e v dual proxy režime.

Aktuálna inicializácia SIRUP-ov po ich spustení prebieha nasledovne: Na začiatku Alica posiela REGISTER správu na server Kamailio. LAN SIRUP je pritom nastavený ako outbound proxy, a teda REGISTER zachytí. Pozmení ho a preposiela na adresu WAN SIRUP-u, ktorú sme vopred špecifikovali v konfiguračnom súbore. Ten po prijatí prvej SIP správy zistí skutočnú externú adresu LAN SIRUP-u, ktorú si uloží a všetky nasledujúce správy smeruje cez túto externú adresu. Pokým WAN SIRUP neprijal REGISTER, nemôže žiadne správy smerovať, nakoľko nepozná externú adresu LAN SIRUP-u.

Po novom predpokladáme, že každý klient bude mať u seba jednu bežiacu inštanciu SIRUP-u. Na druhej strane sa bude nachádzať poskytovateľova inštancia serverového SIRUP-u, ktorá bude bežať na viacerých portoch a pomocou MPLS tunelov na základe portov vyberať cestu pre reláciu.

Aktuálne je SIRUP riešený ako jednoduchý preposielač. Všeobecný postup spracovania paketov sa dá popísať takto: Po prijatí paketu ho najskôr prezeráme na kernelovej vrstve pomocou nástroja iptables. Kontrolujeme, či paket prišiel z očakávanej adresy a portu (zistené z SDP zo zachytených paketov) a v prípade, že paket dané pravidlo spĺňa, označkujeme ho a prepošleme ďalej na adresu druhej inštancie SIRUP-u. Ďalším pravidlom sa pozeráme do paketu za UDP hlavičku a v prípade, že sa tu nachádza očakávaná značka, odstránime ju a paket prepošleme podľa pravidla ďalej. Táto kontrola značky však nezohľadňuje zdrojovú adresu a port paketu, a preto zbytočne kontroluje reťazec za UDP aj pre pakety bez značky. Z hľadiska efektivity toto riešenie nie je úplne vhodné a zrejme bude potrebné do pravidla zakomponovať aj kontrolu adresy a portu.

Zjednodušená verzia iptables pravidiel:

- očakávaná zdrojová adresa, port (paket neprišiel od druhej inštancie SIRUP-u) -> vlož značku pre daný hovor, pošli na druhú inštanciu SIRUP-u
- očakávaná značka za UDP (paket prišiel od druhej inštancie SIRUP-u) -> odstráň značku a pošli na danú adresu a port podľa mapovania

3.3.4 Prepínanie medzi cestami

Na riešenie problému prehadzovania ciest medzi SIRUP-mi bude potrebné vyriešiť akým spôsobom budú medzi sebou komunikovať klientská a serverová inštancia SIRUP-u. Pokiaľ totiž uvažujeme najmenej vhodné mapovanie pre prechod cez NAPT, ktoré je závislé od cieľovej adresy aj portu, môžeme predpokladať, že keď klient začne komunikovať so serverom na inom porte, zmení sa mu mapovanie a bude vystupovať pod zmeneným portom alebo aj zmenenou adresou. Jedným z riešení by mohlo byť poslanie správy REGISTER od klienta serveru, pomocou ktorej by oznámil svoju novú polohu (IP adresu a port).

Zložitejšia situácia nastáva, ak chceme zmeniť cestu počas prebiehajúceho hovoru. Táto zmena musí byť plynulá a v ideálnom prípade nezaznamenaná komunikujúcimi stranami. Je preto potrebné navrhnúť mechanizmus potvrdzovania zmeny kontaktných informácií SIRUP-ov tak, aby počas hovoru nedošlo k strate paketov. Vychádzajme z predpokladu, že klientský SIRUP posiela správu REGISTER na oznámenie svojej novej externej adresy a portu. Keď túto správu zachytí server, pripraví si potrebné pravidlá a posiela odpoveď 200 OK. Vďaka tejto odpovedi klient vie, že server je pripravený preposielať správy na inom porte a teda môže cieľový port odosielaných paketov zmeniť. Otvorená však zostáva otázka, akým spôsobom je možné potvrdiť správu 200 OK tak, aby klient vedel svoju pripravenosť oznámiť serveru. Správa ACK sa môže použiť len v rámci metódy INVITE a správa PRACK sa používa na potvrdzovanie provizionálnych odpovedí 101-199.

Nakoľko doposiaľ klient aj server boli nastavení tak, aby oba pracovali len s jednou inštanciou, je tiež potrebné zaviesť nejaký identifikátor klientov, pomocou ktorého by sa vedeli prihlásiť voči serveru. Na zabezpečenie jedinečnosti identifikátora ho bude spravovať jeden centrálny uzol.

3.3.5 SIRUP Master

Sirup Master je server ktorý primárne komunikuje s klientskymi (LAN) SIRUP-mi a tvorí riadiacu časť architektúry. Tento server bude mat statickú IP adresu a bude nevyhnutný pre zriadenie spojenia medzi klientovou a serverovou časťou SIRUP architektúry. Keďže SIRUP v súčasnej implementácií dokáže pracovať s protokolom SIP, najrozumnejším riešením bude, aby tento server rozumel tomuto protokolu tiež. Počas štúdia na bakalárskom stupni na predmete Konvergencia mobilných a pevných sietí zadaním bolo naprogramovať vlastné SIP proxy, ktoré ovláda základné SIP správy. Tento proxy server môžeme využiť v tejto architektúre. Server bude tvoriť rozhranie medzi manažmentom a SIRUP architektúrou, bude zbierať informácie od klientov a oznamovať im, čo robiť v prípade, keď dôjde k nepredvídanej situácie na základe pokynov manažment servera.

Server s klientmi SIRUP-u bude komunikovať pomocou protokolu SIP, manažérskym serverom pomocou websocket-ov v JSON objektoch. Pre rýchlejšiu komunikáciu si bude udržiavať vlastnú databázu klientov a serverov architektúry SIRUP.

3.4 Manažment Server

Manažmentový server je ústredný bod celej architektúry. Mal by zabezpečovať komunikačné rozhranie medzi všetkými uzlami infraštruktúry. Takéto jednotné rozhranie sa nazýva API.

3.4.1 Implementačné nástroje

3.4.1.1 Programovací jazyk JavaScript

JavaScript je, ako už názov napovedá, skriptovací objektovo orientovaný programovací jazyk, ktorý sa hlavne využíva v moderných webových prehliadačoch ako nástroj interakcie s používateľom. Syntax je ovplyvnený jazykom C a veľa názvov je prebraných s jazyku Java. Navzdory svojmu názvu, má ale JavaScript veľmi málo spoločné s jazykom Java. JavaScript bol prvý krát štandardizovaný v roku 1997. Momentálne posledná verzia štandardu 5.1 je z roku 2011[1].

Väčšinou sa k jazyku JavaScript pristupuje interpretovaným spôsobom. To znamená, že zdrojový kód sa priamo vykonáva bez predchádzajúceho prekladu do strojového kódu. Engine V8 od Google si ale zvolil iný prístup. Zdrojový kód sa kompiluje priamo do natívneho kódu použitej architektúry. Podporované sú 32 a 64 bitové verzie x86, ARM a MIPS, čo zahrňuje majoritnú väčšinu bežne používaných systémov. Takýto prístup dovoľuje výrazne optimalizovať vykonávajúci sa kód a zvyšuje výkon aplikácií vykonávaných na tomto engine [2][3].

Veľa ľudí jazyk JavaScript škatuľkuje len do sveta webových prehliadačov, s tým že to nie je plnohodnotný jazyk. To nie je tak úplne pravda, pretože existuje viacero implementácií mimo webových prehliadačov, napríklad platforma Node.js, určená pre vývoj webových aplikácií.

3.4.1.2 Platforma Node.js

Node.js je platforma postavená na Chrome JavaScript engine pre jednoduché vytváranie rýchlych a škálovateľných sieťových aplikácií. Platforma Node.js vznikla v roku 2009 a relatívne rýchlo nadobudla značnú popularitu, nielen v komunite vývojárov, ale aj medzi veľkými softvérovými hráčmi, ako napríklad Microsoft. Je často nazývaná ako serverový JavaScript. Základom tejto platformy je JavaScript engine V8 od spoločnosti Google, ktorý je voľne dostupný pod BSD licenciou s otvoreným zdrojovým kódom. Tento engine je zodpovedný za vykonávanie JavaScript kódu v prehliadači Chrome od spoločnosti Google [2].

Node.js implementuje programovací model udalostí postavený na neblokujúcich vstupno-výstupných operáciách. Väčšina bežných serverových implementácií vytvára pre každého klienta samostatné vlákno v systéme. Node.js serverová aplikácia beží a obsluhuje klientov v jednom vlákne, čo znamená, že odpadáva záťaž systému pri prepínaní medzi množstvom procesov. Tieto vlastnosti robia túto platformu ideálnu pre aplikácie v reálnom čase, ktoré využívajú veľké množstvo vstupno-výstupných operácií od množstva klientov [2].

Z hľadiska programátora je Node.js taktiež zaujímavá platforma, pretože aplikácie sa píšu v jazyku JavaScript, ktorého aspoň základy ovláda veľké množstvo webových vývojárov. Jednou z nevýhod je, že väčšinu kódu tvoria asynchrónne JavaScript funkcie, ktoré pri bežnom zápise vytvárajú relatívne neprehľadnú štruktúru kódu a sťažujú jeho čitateľnosť [4].

Výhodou je modulový systém, ktorý zjednodušuje znovupoužitelnosť jednotlivých častí a výrazným spôsobom urýchľuje vývoj. Napríklad pri použití modulu s názvom Express, je implementácia webového servera záležitosť niekoľko málo riadkov kódu. Takýto typ systému dovoľuje programátorovi sústrediť sa na samotné správanie sa vyvíjanej aplikácie. Na manažment použitých modulov sa používa program npm, ktorý sa stará napríklad o inštaláciu aktualizáciu modulov [5].

Node.js je možné inštalovať a prevádzkovať na systémoch s Windows, Linux alebo OS X. Niektoré moduly ale nemusia podporovať všetky tri platformy, no multiplatformovosť je značnou výhodou [2].

Všetky tieto vlastnosti Node.js kvalifikujú túto platformu ako ideálnu pre implementáciu manažmentového servera a pre vytvorenie API rozhrania medzi jednotlivými časťami navrhovaného systému. Počíta sa s obsluhou veľkého množstva požiadaviek, ktoré ale sú nenáročné na výpočtový výkon, čo je dôležité z výkonnostného hľadiska [6].

3.4.1.3 Databáza

Pri použití Node.js ako platformy pre manažérsky systém, je okruh možných databáz veľmi široký. Je možné použitie klasických SQL databáz ako napríklad MySQL alebo PostgreSQL. No lepšie výsledky je možné dosiahnuť pri použití NoSQL databáz, ktoré sú svojou podstatou bližšie k tejto platforme, pretože poväčšine dokážu pracovať priamo s JavaScript objektmi vo formáte JSON. Vhodné sú hlavne dve databázy s názvom MongoDB a CouchDB. Obe tieto databázy môžu bežať pod systémami Windows, Linux alebo OS X [7].

3.4.1.4 Mongo Database

MongoDB sa svojím prístupom podobá na tradičné SQL databázy ako MySQL alebo PostgreSQL. Prístupy k dátam sú definované cez indexy, podobne ako v SQL

databázach. Použitie tejto databázy je vhodné, ak je potrebné mať dynamické dopyty a vysoká rýchlosť je zásadnou podmienkou. Pre komunikáciu s touto databázou sa používa vlastný binárny protokol. Databáza je dostupná pod AGPL licenciou [8].

3.4.1.5 Couch Database

Existuje aj NoSQL databáza, ktorá ale na rozdiel od MongoDB používa namiesto dynamických dopytov MapReduce funkcie. Je zameraná na jednoduché použitie a zaručenie konzistencie dát. Použitie tejto databázy je vhodné, ak sa dáta v nej akumulujú a príliš často nemenia. Využívajú sa dopredu definované dopyty. Komunikáciu je zabezpečená cez REST API, a teda protokol HTTP. Databáza je dostupná pod Apache licenciou [9].

3.5 Manažment API

3.5.1 API

Aplication programming interface (API) je špecifikácia prístupu k jednotlivým funkciám programových celkov. Väčšinou je to zbierka procedúr a funkcií, ktoré sú poskytnuté používateľovi pre prístup ku knižnici alebo službe a definujú jej vonkajšie správanie sa.

3.5.2 JSON

JavaScript Object Notation (JSON) je otvorený štandardizovaný formát používaný primárne na prenos dát. JSON je definovaný v štandarde RFC 4627. Formát je odvodený z JavaScript, no je jazykovo nezávislý a dostupný v množstve iných jazykov.

Podobným štandardom ako JSON je XML. Oproti XML má JSON dátovo úspornejšiu notáciu. Je natívne podporovaný jazykom JavaScript, čo je výhodné v spolupráci s Node.js. Pre JSON sú charakteristické tieto dve vlastnosti:

- Notácia je tvorená kolekciami kľúč - hodnota podobne ako v iných jazykoch sú objekt, štruktúry alebo hash tabuľky.
- Hodnoty tvoria usporiadaný zoznam podobne ako polia, alebo usporiadané listy v iných jazykoch.

Tieto dva kľúčové prvky zabezpečujú vysokú prenosnosť medzi rôznymi jazykmi, rýchlosť spracovania a nenáročnosť na veľkosť a výkon. Pre všetky tieto vlastnosti je JSON ideálny formát na prenos dát medzi klientmi a serverom v budovanej manažment API [10].

3.5.3 Implementácia API

K vytváraniu API je možné pristupovať rôznymi spôsobmi. Najtradičnejšie je použitie Representational state transfer (REST) architektúry. Pre aplikácie v reálnom

čase sa stáva obľúbená relatívne nová platforma Socket.io, ktorej serverová časť je implementovaná v prostredí Node.js.

3.5.3.1 REST architektúra

REST je architektúra, nie samotná implementácia. To znamená, že je nezávislá od platformy a protokolov a sústredí sa na samotné prvky tejto architektúry. Vo svete Internetu je to momentálne dominantná API architektúra a väčšinou využíva protokol HTTP, s ktorým má veľa spoločného. Pri využití HTTP sa implementácia nazýva RESTful [11].

Architektúra je postavená na princípe klient - server a využíva sa metóda požiadavka - odpoveď. Podobne ako v protokole HTTP, najviac používané požiadavky sú:

- **GET** - Požiadanie o dáta
- **POST** - Požiadavka o modifikáciu dát
- **PUT** - Požiadavka na vytvorenie alebo prepísanie dát
- **DELETE** - Požiadavka na vymazanie dát

Samotné dáta v požiadavkách a odpovediach bývajú často objekty typu XML alebo JSON.

3.5.3.2 Socket.io knižnica

Socket.io je JavaScript knižnica implementovaná na platforme Node.js. Používa sa pre komunikáciu medzi serverom a klientmi v reálnom čase. Primárna komunikačná technológia je WebSocket protokol, no táto knižnica od neho nie je závislá a v prípade jeho nedostupnosti na používanej platforme používa iné protokoly a ich implementácie ako Flash Sockets, JSONP pooling alebo AJAX long pooling.

Na serverovej strane spolupracuje s jednoduchým HTTP serverom v prostredí Node.js, či už vstavaným alebo o funkcie bohatší modulom Express. Serverová časť ponúka asynchrónne posielanie správ klientom, a takisto asynchrónne prijatie správ od klientov. Takéto správy môžu obsahovať dátové objekty. Zaujímavá je aj schopnosť jednoduchého rozosielania správ všetkým klientom naraz.

Klientska časť býva často webový prehliadač a klient teda býva napísaný v jazyku JavaScript. No nemusí to tak byť a klientska aplikácia je nezávislá na platforme a existujú implementácie pre väčšine bežne rozšírených jazykoch ako C, C++, C#, Java atď. Aplikačné rozhranie u klienta je väčšinou totožné ako u serveru, no záleží od konkrétnej implementácie [12].

3.6 Manažment užívateľského rozhrania

Dôležitým prvkom každej aplikácie je ako pôsobí a vyzerá navonok. Ako sa s ňou narába a aké je rozhranie medzi používateľom a aplikáciou. Rozloženie jednotlivých prvkov na obrazovke by malo byť intuitívne a prehľadné. V tejto časti sa budeme zaoberať užívateľským rozhraním - frontendom pre zobrazenie informácií potrebných na riadenie systému.

Frontend je rozhranie medzi používateľom a vnútornou funkčnou časťou aplikácie. Slúži na zber vstupných údajov v rôznych formách od používateľa alebo na zobrazenie výstupných informácií systému.

Naša aplikácia má umožniť **aktívny manažment** relácií a mala by byť schopná adaptovať sa na náhle zmeny v sieti. Aby táto adaptácia bola prezentovaná klientovi je potrebné zobrazovať niekoľko dôležitých informácií. Tieto informácie upozornia používateľa na zmeny, ktoré aplikácia vykonala a prečo.

Aby sme dosiahli výnimočné zobrazenie všetkých relevantných informácií je potrebné analyzovať dostupné nástroje na zobrazovanie. Či už formou grafu alebo pomocou dynamických prvkov, ktoré poskytujú jednotlivé nástroje rozhrania API. V celom manažmente budeme používať **skriptovací programovací jazyk JavaScript** s rôznymi platformami a knižnicami, či už štandard JSON alebo Socket.io prípadne Node.js, je preto vhodné použiť aj na tvorbu rozhrania medzi klientom a manažmentom **JavaScript**. Zaručuje nám to nielen kompatibilitu jednotlivých programových celkov, ale aj zjednodušenie práce z dôvodu lepších znalostí daného jazyka. Ďalšou výhodou použitia je **podpora väčšiny prehliadačov** bez potreby inštalovať zložité a rozsiahle dodatočné nástroje.

Pri tvorbe webových stránok či aplikácií netreba zabúdať aj na **hypertextový značkovací jazyk HTML**. Je určený na tvorbu webových stránok a zobrazovanie informácií vo webových prehliadačoch. Poslednou verziou je HTML 5, ktorá kladie dôraz na rýchlejší a jednoduchší zápis značiek a zároveň na ich účinnosť. Podporuje tiež vytváranie aplikácií, ktoré fungujú bez internetového pripojenia a ukladajú dáta do lokálneho úložiska na počítači užívateľa. Obsahuje množstvo ďalších užitočných prvkov a značiek ako je podpora 2D kreslenia alebo prehrávanie audio a video súborov.

Teraz sa pozrieme na informácie, ktoré budeme zobrazovať:

- **RTT – round-trip delay time** – čas potrebný na prenesenie paketu zo zdrojového uzla do cieľovej siete
- **PL – packet loss** – strata paketov počas prenosu
- **Jitter** – kolísanie veľkosti oneskorenia paketu pri prechode sieťou
- **Bandwith** – aktuálna šírka pásma

- **Štatistické informácie** – počet hovorov na danej linke, počet registrovaných klientov, počet relácii na jedného klienta, počet prenesených dát na jedného klienta, počet pádov linky
- **Informácie o klientoch** – IP adresa a port klienta, užívateľské meno
- **Informácie o prepínaní liniek** – počet prepnutí, ktorá relácia bola prepnutá a kam
- **Indikácia pádu linky**

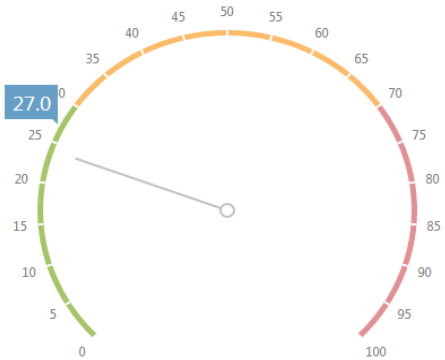
Všetky informácie, ktoré sme vymenovali by bolo najvhodnejšie zobrazovať formou grafov využitím knižníc jazyka JavaScript alebo použitím jazyka HTML. Kombináciou týchto dvoch programovacích jazykov na tvorbu webových aplikácií dosiahneme, aby aplikácia manažmentu spĺňala a zobrazovala všetko potrebné v dokonalom grafickom prevedení. Aby sme dosiahli želaný efekt, pozrieme sa bližšie na jednotlivé knižnice v programovacom jazyku JavaScript pre zobrazovanie informácií.

Hlavným prvkom webového rozhrania budú grafy. Existuje veľké množstvo knižníc v jazyku JavaScript, ktoré podporujú tvorbu grafov. Rozdiely medzi knižnicami sú len v typoch grafov, ktoré knižnice dokážu zobraziť. Základné porovnanie tých najlepších knižníc na zobrazenie grafov je na obrázku číslo 4.

	Supported Chart Types											
	Line	Timeline	Scatter	Area	Pie	Rectangular Pie	Donut	Bullet	Radar	Funnel	Gantt	
												Grouped
amCharts	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes
CanvasJS	Yes	No	Yes	Yes	Yes	No	Yes	No	No	No	No	No
ChartJS	Yes	No	Yes	Yes	Yes	No	Yes	No	No	No	No	Yes
FusionCharts	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes

Obrázok 3-4 Porovnanie knižníc na zobrazenie grafov[14]

Podľa nasledujúceho porovnania by sa mohlo zdať, že najlepšou knižnicou na použitie je *FusionCharts*. No najväčším hendikepom tejto knižnice je, že **nepodporuje Gauge Widgets**. Ukážku takéhoto grafu môžeme vidieť na obrázku 5. Preto sme sa rozhodli, že budeme používať knižnicu **ChartJS Dev Express**, ktorá má množstvo ďalších výhod, ktoré popíšem v priebehu tejto kapitoly.

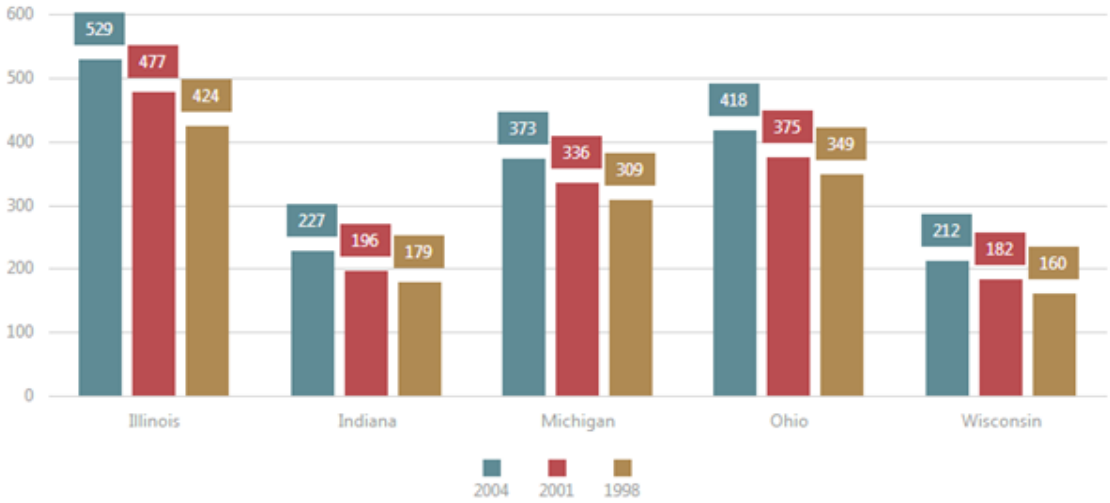


Obrázok 3-5 Graf HTML JS Gauge Widgets[13]

Tento graf **je najlepší** na zobrazovanie informácií, ktoré potrebujeme. Jedná sa hlavne o oneskorenie prenosu paketov (RTT), strata paketov (PL), odchýlka oneskorenia paketu (Jitter) a šírka pásma (Bandwith). Po nadviazaní spojenia s manažmentom, ktorý bude mať všetky potrebné informácie k dispozícii, budú posielané do webovej aplikácie periodicky. To nám zabezpečí, že graf bude mať vždy aktuálne informácie, ktoré bude na takomto grafe zobrazovať.

Zobrazovanie štatistických informácií by bolo vhodné rozdeliť. A to konkrétne na **historické informácie**, teda tie ktoré boli zaznamenané niekoľko dní dozadu **a na aktuálne**, ktoré sú v danej chvíli. Na zobrazenie aktuálnych informácií napríklad o počte klientov na danej linke môžeme použiť graf na obrázku 6, pre zobrazenie historických informácií by postačoval graf na obrázku 7.

Obrázok 3-6 Graf pre zobrazovanie aktuálnych štatistických informácii[13]



Obrázok 3-7 Graf pre zobrazovanie historických štatistických informácii[13]

Podobné grafy by mohli byť použité aj na zobrazovanie ďalších informácií. Implementácia je pomerne jednoduchá a veľkou výhodou je, že knižnicu ChartJS podporuje mnoho mobilných platforiem ako Android, iOS, Windows Phone a taktiež všetky mobilné aj desktopové webové prehliadače. Medzi ďalšie výhody tejto knižnice môžeme zaradiť[13]:

- *interakcia grafov – pri označení určitej časti grafu sa aktualizujú aj grafy s ním spojené,*
- *jednoduchá možnosť priamej konfigurácie,*
- *prispôsobenie sa väčšine mobilných zariadení,*

- *obrovské množstvo rôznych druhov grafov a nastavení,*
- *možnosť spájať rôzne typy grafov do jedného,*
- *podpora grafov s výberom úseku.*

Ďalšou veľkou výhodou tejto knižnice je *podpora monitorovacích funkcií.* Obsahuje možnosť vytvoriť si monitorovací panel, kde pomocou kombinácie rôznych grafov a vhodného prostredia dokážeme monitorovať rôzne prvky našej siete. Ukážka panelu je na obrázku 8.

Obrázok 3-8 Monitorovací panel[13]

Ako už bolo spomenuté, knižnica ktorú sme opisovali spolu s jazykom HTML 5 sú najvhodnejšou voľbou pre potreby monitoringu našej VoIP siete a dokonale sedia na požiadavky zobrazovania potrebných informácií.

3.7 Simulácia siete

Aby bolo možné overiť funkčnosť implementovaného riešenia, je potrebné vytvoriť prostredie pre jeho testovanie. Vytvorenie sieťových tunelov na reálnych alebo virtuálnych zariadeniach by prestavovalo zvýšené nároky na počet a výkon použitých zariadení, a tiež by obmedzilo možnosti nútenej zmeny charakteristiky tunelov. Z pohľadu manažéra relácií sú sieťové tunely opísané určitými parametrami a ich skutočná existencia nie je potrebná. Cieľom je možnosť meniť parametre opisujúce tunely tak, aby výsledná aplikácia preukázala schopnosť adaptovať sa na tieto zmeny.

3.7.1 Network Emulation (NetEm)

NetEm (Network Emulation) poskytuje potrebnú funkcionalitu pre testovanie protokolov imitovaním vlastností skutočných sietí. Aktuálna verzia umožňuje vytvorenie oneskorenia, straty, duplicity, chybovosti a preusporiadania paketov. Na obmedzenie šírky pásma možno použiť Token Bucket Filter (TBF). Súčasné verzie Linuxového jadra (od verzie 2.6) obsahujú podporu pre tieto nástroje dostupné v rámci balíčka iproute2. NetEm sa ovláda pomocou konzolového programu ‘tc’. Syntax príkazov je pomerne komplikovaná a konfigurácia si vyžaduje hlbšie pochopenie problematiky. Preto je

vhodné navrhnúť rozhranie, ktoré poskytne používateľsky prívetivejšie prostredie s podporou transformovania vstupov do formy vyžadovanej programom tc[15].

Pakety je možné klasifikovať pomocou filtrov. Na účely zadania je vhodné použiť filtrovanie podľa čísla UDP portov paketov, keďže tieto charakterizujú, ktorým tunelom má prebiehať komunikácia. Rozdelenie do skupín umožňuje definovať odlišné parametre preposielania pre rôzne tunely.

3.7.2 Informácie o kvalite linky

1. **Round – Trip Time (RTT)**

Inak nazývaný aj **round-trip delay time (RTD)** je čas potrebný na cestovanie paketu medzi zdrojovým uzlom v sieti a konečnou stanicou. V kontexte počítačových sietí je možné tento čas zistiť zadaním príkazu *ping* na zdrojovom počítači s IP adresou cieľového. Nazýva sa aj *ping time*. Jeho veľkosť závisí od rôznych faktorov:

- rýchlosť internetového prenosu dát zdrojového zariadenia,
- typ prenosového média,
- fyzická vzdialenosť medzi zdrojom a cieľom,
- počet uzlov v sieti,
- množstvo dopravy na sieti atď.

RTT je aj jedným z mnohých prvkov, ktoré ovplyvňujú oneskorenie siete, čo je čas medzi vytvorením požiadavky na dáta a úplným vrátením alebo zobrazením týchto súborov. Čas na prenesenie paketu je možné aj teoreticky veľmi ľahko vypočítať, no bude to len orientačný výsledok a my potrebujeme pracovať s presnými číslami.

Meranie RTT pomocou ICMP paketov alebo tzv. pingu, nie vždy dáva relevantné výsledky v reálnych sieťach. Existuje ešte jeden spôsob merania oneskorenia paketov prostredníctvom nástroja s názvom **„TCP Ping“.** Myšlienka tohto nástroja je rovnaká ako pri normálnom pingu, ale namiesto používania ICMP paketov, odosiela TCP SYN pakety do cieľového zariadenia a meria čas medzi subsekvenciou prijatia **SYN/ACK alebo RST**. Nástroj beží pod operačným systémom Linux a obsahuje niekoľko prepínačov na nastavenie portu alebo rôznych módov. Obrázok 9 ukazuje fungovanie takéhoto nástroja na reálnom stroji.

```
[6:22pm] luthien:~/proj/tcpping-% sudo ./tcpping sumatra
TCP PING sumatra.internal.kehlet.cx (10.16.74.2:80) on en1
SYN/ACK from 10.16.74.2: seq=1 ttl=64 time=1.047ms
SYN/ACK from 10.16.74.2: seq=2 ttl=64 time=0.965ms
SYN/ACK from 10.16.74.2: seq=3 ttl=64 time=1.081ms
SYN/ACK from 10.16.74.2: seq=4 ttl=64 time=1.245ms
^C
--- sumatra.internal.kehlet.cx TCP ping statistics ---
4 SYN packets transmitted, 4 SYN/ACKs and 0 RSTs received, 0.0% packet loss
round-trip min/avg/max = 0.965/1.084/1.245 ms
```

Obrázok 3-9 Nástroj TCP Ping

Zdroj: http://www.kehlet.cx/articles/77.html

2. **Packet Loss (PL)**

K strate paketov v počítačovej sieti dochádza v prípade, že prenášaný paket nedosiahne svoj cieľ. Táto vlastnosť je taktiež v počítačových sieťach dôležitá a je jednou z hlavných faktorov, ktoré sa zisťujú pri každej prevádzke. Stratu paketu môže spôsobiť niekoľko faktorov, no takým najčastejším je degradácie signálu pri prechode cez sieťové médium, resp. zahltenie linky. Potom to môžu byť poškodené pakety, chybný sieťový hardvér, chyba sieťového ovládača alebo chyby v smerovaní.

3. **Jitter**

V počítačových sieťach vyjadruje kolísanie oneskorenia príchodu paketov. Vzniká na smerovačoch ako dôsledok funkcie interných prioritných radov smerovačov a iných sieťových zariadení, a tiež smerovacích zmien. Hodnotu kolísania oneskorenia je možné zistiť opakovaným meraním oneskorenia.

4. **Bandwidth**

Prenosová rýchlosť udáva, aký objem informácií sa prenesie za jednotku času. Základná jednotka prenosovej rýchlosti je bit za sekundu. Meranie maximálnej prenosovej rýchlosti sa vykonáva meraním času potrebného na prenesenie daného objemu dát. Možno tiež merať aktuálnu prenosovú rýchlosť. Tá je vyjadrená súčtom veľkostí prenesených rámcov za jednotku času.

4 Špecifikácia

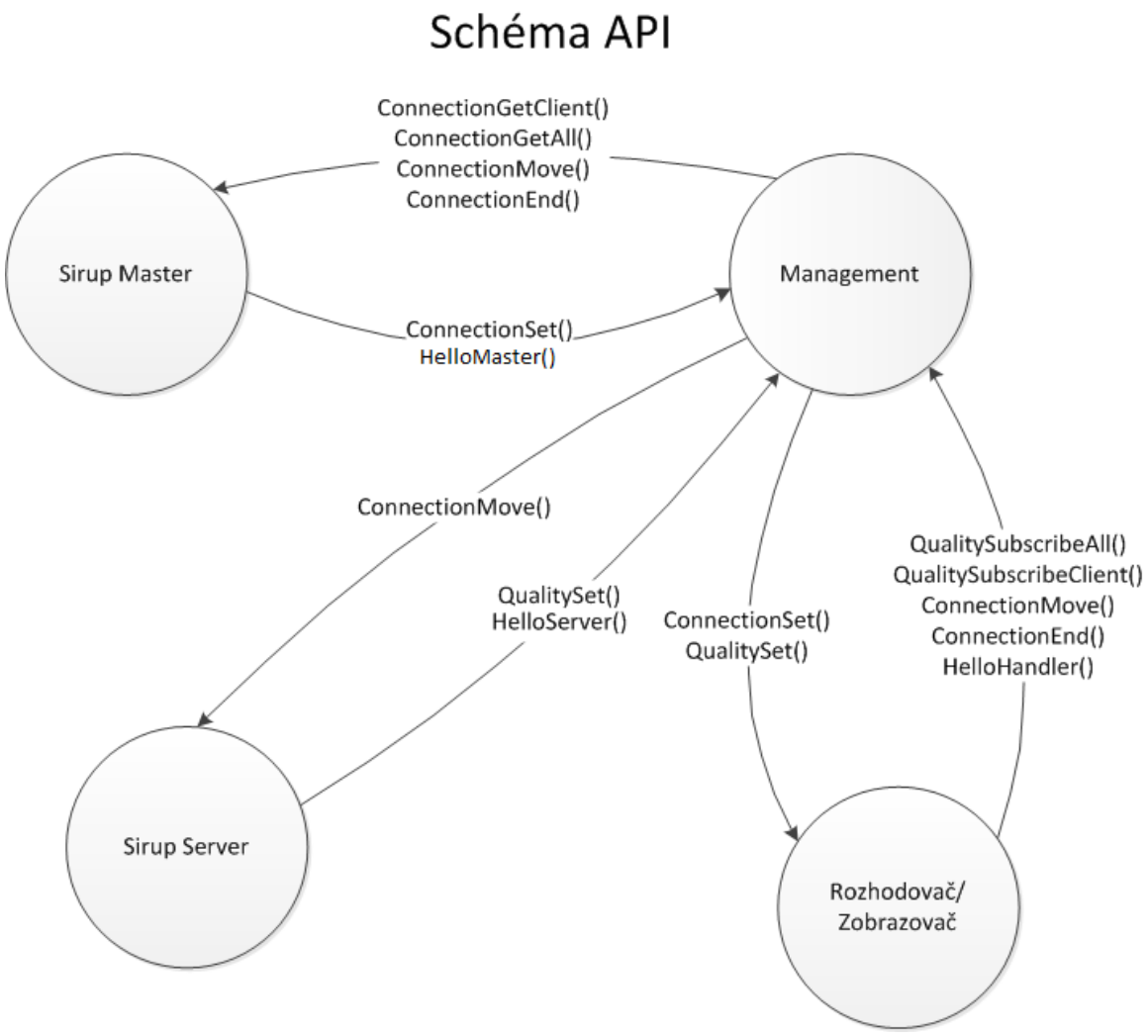
4.1 Funkcie systému

1. Rozkladanie záťaže
2. Monitorovanie multimediálnych relácií
3. Presun multimediálnych relácií medzi serverovými uzlami
4. Vykresľovanie štatistických informácií
5. Rušenie multimediálnych relácií
6. Zabezpečovanie kvality relácií - agresívnejšie prepínanie liniek - “platinum / VIP smerovanie”

4.2 Údaje

Dáta prenášané a archivované v databáze budú reprezentované ako JSON objekty.

4.3 Manažment API správy



Obrázok 4-1 Tok správ medzi uzlami systému

Správy o kvalite linky:

- **QualitySet()**
  Správa obsahujúca informácie o kvalite linky.
- **QualitySubscribeClient()**
  Správa žiadajúca o pravidelné preposielanie informácií o kvalite linky jedného klienta.
- **QualitySubscribeAll()**
  Správa žiadajúca o pravidelné preposielanie informácií o kvalite linky všetkých klientov.

Správy spojené so správou klientov

- **ConnectionSet()**
  Vytvorenie nového spojenia.
- **ConnectionGetClient()**
  Získanie informácií o spojení jedného klienta.
- **ConnectionGetAll()**
  Získanie informácií o spojení všetkých klientov.
- **ConnectionMove()**
  Zmena parametrov spojenia.
- **ConnectionEnd()**
  Ukončenie spojenia.

Správy spojene s pripojením uzlov architektúry

- **HelloMaster()**
  Prihlásenie Sirup mastra.
- **HelloServer()**
  Prihlásenie nového server Sirupu.
- **HelloHandler()**
  Prihlásenie zobrazovacej/rozhodovacej jednotky.

4.4 Správanie systému

4.4.1 Manažment

Úlohou manažmentu je poskytovať aplikačné rozhranie pre komunikáciu uzlov architektúry. To znamená, že musí vedieť posielať a prijímať správy, ktoré sú špecifikované v pasáži manažment API správy.

Ďalšou úlohou manažmentu je archivovať relevantné údaje v správach do databázy, a tiež si ich z tejto databázy vedieť vyžiadať.

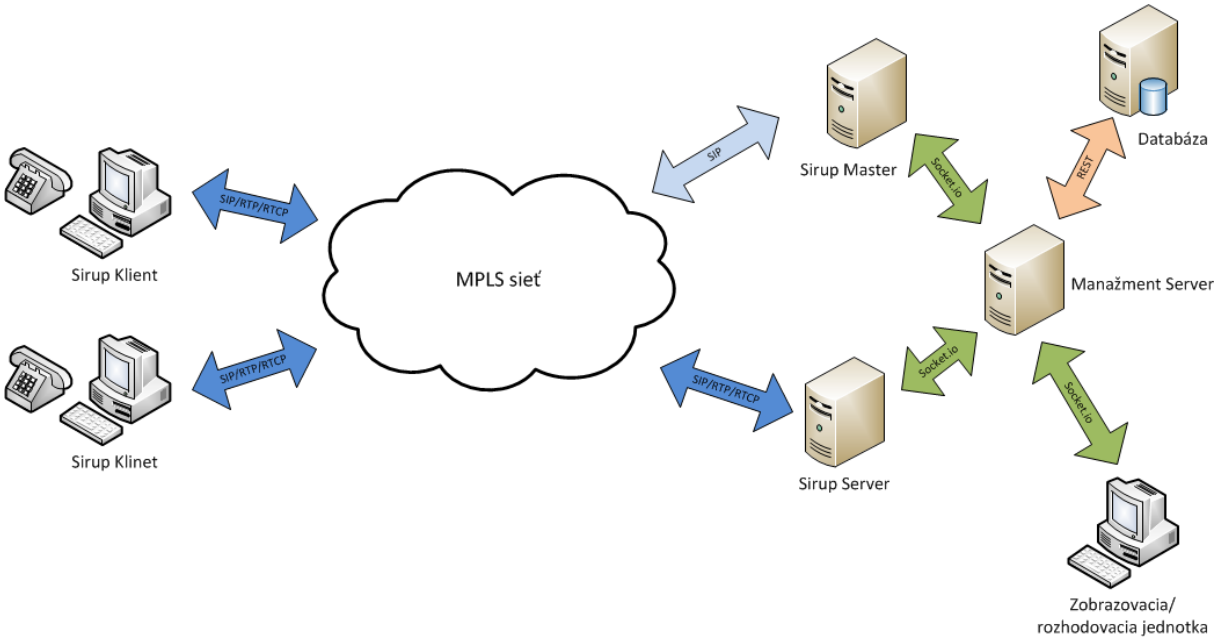
4.4.2 SIRUP komunikácia

- SIRUP-y komunikujú navzájom výlučne pomocou SIP správ

- Manažment nekomunikuje s klientmi priamo
- Klientske SIRUP-y majú pridelený jedinečný identifikátor
- Klientska inštancia SIRUP-u oznamuje svoju polohu voči serverovej pomocou správy REGISTER, obsahujúcej jej identifikátor
- Klientska inštancia SIRUP-u komunikuje s riadiacou inštanciou SIRUP-u pomocou správy SUBSCRIBE
- Riadiaca inštancia SIRUP-u udržiava s klientskou inštanciou reláciu pomocou správ NOTIFY

5 Návrh

5.1 Architektúra



Obrázok 5-1 Architektúra systému

5.2 Manažment API

Navrhované riešenie využije platformu Socket.io, ktorá bola v časti analýzy porovnávaná s bežne používanou REST architektúrou. Oproti REST architektúre sa správy Socket.io nemusia spoliehať na schému požiadavka - odpoveď. Správy môžu byť posielané kontinuálne bez nutnosti žiadať o ne. Táto schopnosť pomáha pri implementácií architektúry, ktorá má pracovať v reálnom čase a je pre ňu dôležitý faktor čas odozvy.

Ďalšou výhodou oproti REST architektúre je že vzťahy medzi uzlami nemusia byť striktne klient - server. To dáva istú flexibilitu pri návrhu riešenia aplikačného rozhrania, keďže v REST architektúre server nemá možnosť žiadať dáta od klienta.

Všetky tieto fakty hovoria v prospech platformy Socket.io, ktorá by mala byť jednoducho implementovateľná na všetkých uzloch architektúry pomocou dostupných voľne šíriteľných knižníc.

Webové rozhranie medzi klientom a manažmentom bude mať na starosti knižnica ChartJS založená na platforme JavaScript. Zobrazenie potrebných informácii, bude vo forme grafov v spolupráci s hypertextovým značkovacím jazykom HTML.

5.3 Databáza

V časti analýzy sme porovnávali dve databázy MongoDB a CouchDB. Rozhodli sme sa použiť druhú menovanú, pretože predpokladáme, že dáta budeme hlavne archivovať a vykonávať nad nimi len dopredu definované operácie. Dáta by sa nemali príliš často meniť a nebudeme potrebovať ani dynamické dopyty. MongoDB má síce výhodu vysokej rýchlosti, no konzistencia dát nie je zaručená, čo opäť nahráva CouchDB databáze. Obidve databázy ukladajú dáta v tvare JSON objektov, čo je vyhovujúce.

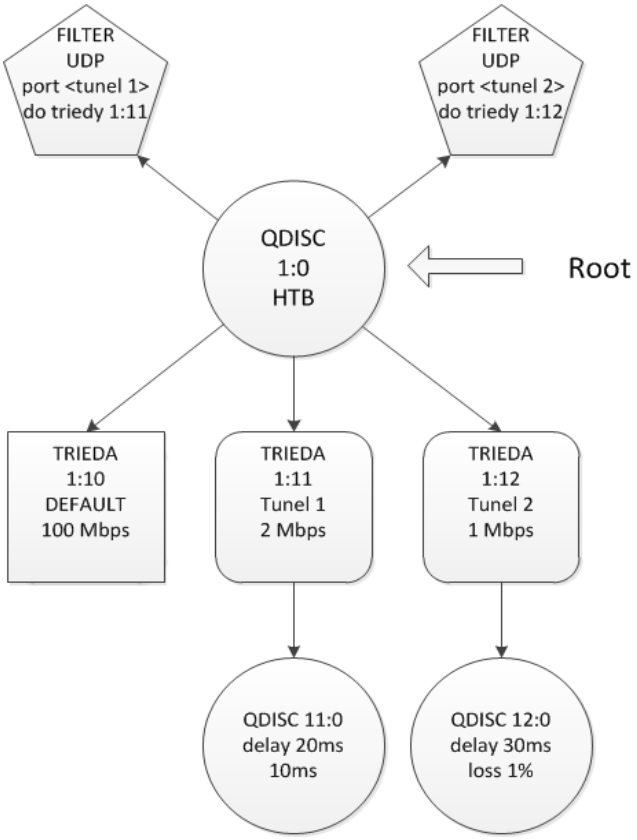
5.4 Údaje

Architektúra nášho systému sa skladá s rôznych systémov a je vhodné použiť spoločný formát pre reprezentáciu údajov. Dáta prenášané a archivované v databáze budú reprezentované ako JSON objekty, ktoré sú podporované na použitých systémoch a natívne podporované manažmentovým systémom a databázou.

5.5 Simulácia siete

Simulátor potrebný pre testovanie funkčnosti systému predstavuje oddelený a samostatne funkčný celok. Poskytuje možnosť nastavenia sieťových parametrov pre jednotlivé tunely a tiež možnosť zobrazenia aktuálne platných nastavení. Zmenou nastavení ovplyvňuje správanie systému, avšak žiadnym spôsobom s ním priamo nekomunikuje. Preto je voľba prostredia a jazyku na implementáciu nezávislá od spôsobu implementácie ostatných častí systému.

Samotné vykonávanie zmien v sieťovej komunikácií vykonáva nástroj NetEm. V predvolenom stave sa odchádzajúce pakety pridávajú do koreňového radu(QDISC root). Aby bolo možné nastavovať parametre pre rôzne tunely vytvorí sa zodpovedajúci počet radov (QDISC) a špecifikujú sa parametre radu, ako napríklad oneskorenie a odchýlka oneskorenia. Pre odchádzajúce pakety je zvolený zodpovedajúci rad podľa triedy, do ktorej patria. Triedy tiež umožňujú obmedzenie šírky pásma pre jednotlivé tunely. Trieda DEFAULT klasifikuje komunikáciu, ktorá nemá prechádzať cez tunely. Príslušnosť k triedam je určená pomocou filtrov. Filter hľadá zhodu v poliach paketu, ktoré označujú typ transportného protokolu a číslo portu. Ak sa nájde zhoda, paket sa zaradí do príslušnej triedy. Ak žiadny z filtrov nenájde zhodu, paket bude zaradený do triedy DEFAULT.



Obrázok 5-2 Štruktúra radov, tried a filtrov

NetEm môže byť nakonfigurovaný zrkadlovo na dvoch sieťových adaptéroch počítača, medzi ktorými bude preposielať komunikáciu. V tomto zapojení budú klientské zariadenia pripojené na jednom sieťovom adaptéri a serverové uzly pripojené na druhom sieťovom adaptéri. Počítač s úlohou preposielača bude simulovať existenciu sieťových tunelov oneskorením preposielania a zmenou ďalších sieťových parametrov. Pre testovanie klientskej aj serverovej časti na jednom zariadení sa použije sieťový adaptér Loopback. Je potrebné uvedomiť si, že pri komunikácií spôsobom otázka - odpoveď prechádzajú sieťou dve správy, pričom parametre siete ovplyvnia najprv otázku a neskôr aj odpoveď. Oneskorenie odpovede je teda z používateľského hľadiska dvojnásobné v porovnaní s údajom oneskorenia nastavenom na sieťovom adaptéri.

Pre lepšiu názornosť a používateľsky prívetivejšie ovládanie sa nástroj NetEm bude ovládať cez webové rozhranie. Úlohou rozhrania je zobraziť aktuálny počet tunelov, parametre tunelov a možnosť zmeny parametrov tunelov. Webové rozhranie získava potrebné informácie z výpisu programu ‘tc’. Tento tiež slúži na zmenu parametrov tunelu.

5.6 SIRUP

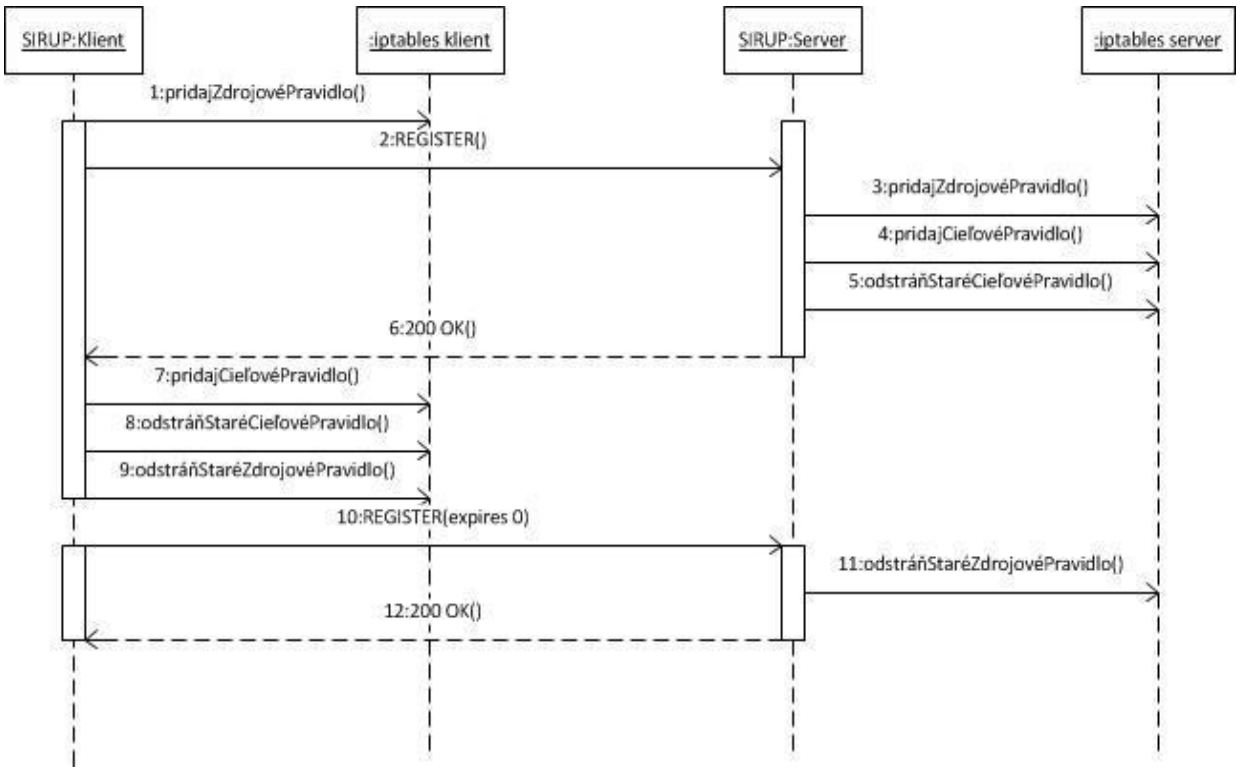
Ako je možné vidieť z architektúry znázornenej na diagrame, predpokladáme, že na našej strane sa bude nachádzať jedna serverová inštancia SIRUP-u (ďalej len server), ktorá bude fungovať na viacerých portoch. Jednotlivé porty pritom určujú MPLS tunely,

cez ktoré vieme viesť jednotlivé relácie. Na opačnej strane tunelov sa budú nachádzať klientské inštancie SIRUP-u (ďalej len klienti). Komunikáciu manažmentovej časti s klientmi bude mať na starosti master SIRUP.

5.7 Klient

Po úspešnom SUBSCRIBE by sa mal klient od mastera pomocou správy NOTIFY dozvedieť o adrese a porte servera, na ktorý sa má zaregistrovať. Tá prebieha štandardne pomocou správy REGISTER, pričom v hlavičke Contact sa musí nachádzať jeho pridelený identifikátor.

Tento jednoduchý REGISTER by postačoval v prípade, kedy aktuálne neprebieha žiaden hovor cez SIRUP-y. Úlohou projektu je však vedieť dynamicky reagovať na zmeny vo vyťažení liniek a na základe nameraných parametrov vedieť plynule presmerovať aktívne hovory cez inú linku. V prípade, že teda manažment rozhodne o jej zmene, posiela klient REGISTER na nový port servera. V tejto fáze už vie z akého portu môže očakávať po novom RTP pakety. Server po prijatí REGISTER správy nastavuje nové pravidlo s klientovou zdrojovou adresou a začína na ňu smerovať RTP pakety. Odpovedá správou 200 OK. Klient teraz môže tiež začať posielať pakety na nový port. Zároveň odstráni staré pravidlo ktoré riešilo preposielanie paketov z pôvodného serverového portu. Aby na serveri nezostávalo staré pravidlo s klientovou pôvodnou adresou a portom posielame REGISTER s Expires hodnotou nastavenou na 0 tak, aby sa z pôvodného portu odhlásil.



Obrázok 5-3 Preregistrácia klienta počas aktívneho hovoru

5.8 Server

Aktuálne je server naprogramovaný len ako jednoduchý preposielač so statickým nastavením klienta. Po novom bude musieť vedieť pracovať na viacerých portoch a tiež si musí vedieť udržiavať asociáciu aktuálne registrovaných klientov s týmito portami.

Po spustení servera posiela správu HelloServer(), ktorou sa registruje voči manažmentu a pomocou nej tiež udržiava spojenie. Ďalšia úloha servera spočíva v meraní kvality linky. Namerané parametre musí pravidelne posielať na manažment pomocou správ QualitySet(). Na základe týchto správ potom manažment rozhodne o prípadnej potrebe zmene linky. Informuje o nej server pomocou správy ConnectionMove().

5.9 SIRUP Master

SIRUP Master bude programovaný v jazyku Java a bude využívať knižnice z vývojovej sady verzie 1.7 s niekoľkými ďalšími knižnicami. Knižnica Jain - SIP verzie 1.2 slúži na rýchle a jednoduché spracovanie SIP správ, json - lib na spracovanie JSON objektov. Pre komunikáciu s websocketmi sme našli viaceré alternatívy ako napríklad jetty - websockets alebo socket.io - java client. Aplikácia si bude udržiavať údaje o klientoch a serverov vo vlastných poliach, ak takéto skladovanie dát nebude postačujúce, použije sa knižnica mysql - connector pre komunikáciu s databázou.

Po naštartovaní, aplikácia otvorí UDP port 5060 na rozhraní, pripojí sa na manažérske API pomocou websocketov a prihlási sa do architektúry so správou HelloMaster(). Z vlastnej databázy načíta aktuálne spojenia alebo si ich vyžiada od manažmentu správou ConnectionGetAll().

Master počúva na porte 5060 a čaká na klientov, aby sa s ním spojili. Ak dostane správu SUBSCRIBE, odpovie klientovi správou 200 OK. Túto informáciu v JSON objekte pošle na API cez websocket v správe ConnectionSet() a čaká na správu ConnectionMove() od manažment serveru. V tejto správe dostane SIRUP Master informáciu, kde sa má klient pripojiť. Túto informáciu pošle klientovi v SIP správe NOTIFY, ktorá sa potvrdzuje správou 200 OK. Správa ConnectionMove() slúži aj na zmenu cieľového bodu počas hovoru, ak dôjde k zahlteniu na niektorom tuneli.

6 Prototyp

6.1 SIRUP

V rámci prototypu SIRUP-ov sa riešil predovšetkým SIRUP - server. Jeho prototyp už funguje na viacerých portoch. Klientom umožňuje registráciu, odregistráciu a vo výslednom efekte preregistráciu. Táto preregistrácia však zatiaľ funguje len mimo aktívneho hovoru, nakoľko v takom prípade je potrebné riešiť aj paralelný prístup klienta na dva porty. Okrem toho bolo uskutočnené nadviazanie základného spojenia medzi klientským SIRUP-om a SIRUP-om Master pomocou správ SUBSCRIBE a NOTIFY.

6.2 SIRUP Master

Statický bod v SIRUP architektúre, pomocou ktorého sa inicializujú spojenia. Klientsky sirup sa po spustení prihlási do siete pomocou správy SUBSCRIBE k tomuto bodu. SIRUP - Master túto informáciu prepošle pomocou JSON objektov k manažment serveru. Manažment server odpovie správou, kde sa ma klient pripojiť. Túto informáciu pošle SIRUP Master klientovi v NOTIFY správe. Klienti pomocou SUBSCRIBE správ oznamujú ich dostupnosť. Správy NOTIFY slúžia na modifikáciu spojenia medzi klientom a serverom.

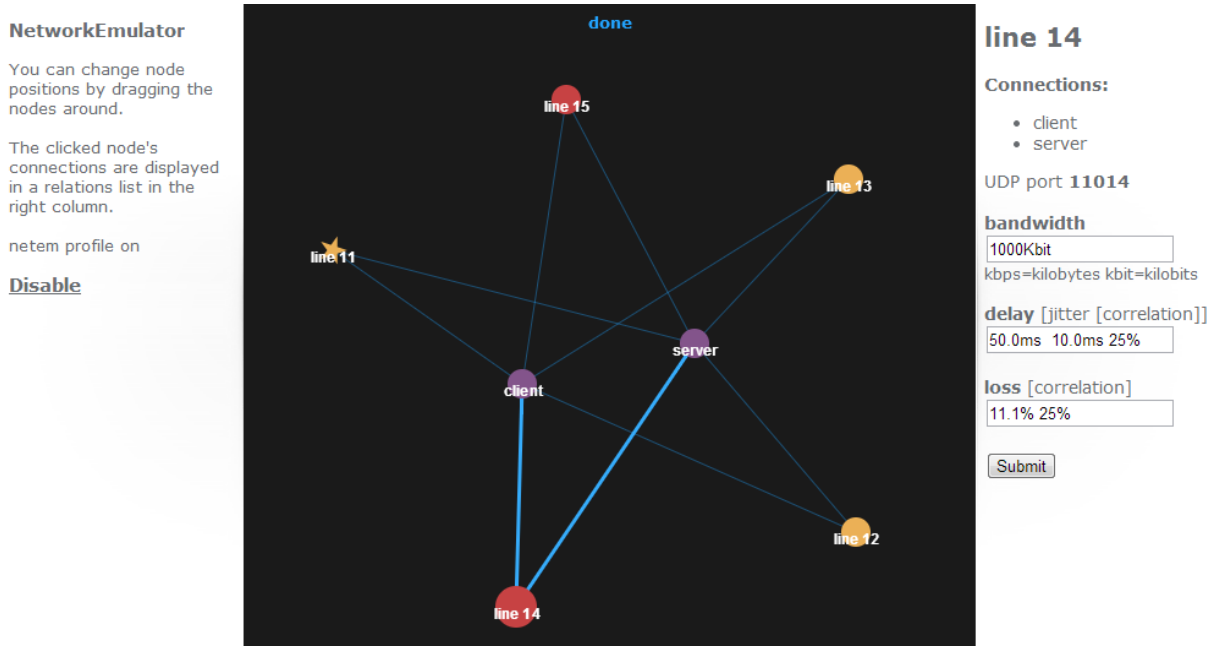
SIRUP - Master dokáže spracovať prichádzajúce správy SUBSCRIBE a na základe časovačov vyhodnotiť dostupnosť klienta. Taktiež dokáže vytvárať a analyzovať JSON objekty, ale nedokáže ich zatiaľ preposlať k manažment serveru. Je schopný tiež vygenerovať náhodnú NOTIFY správu, ktorá zatiaľ neplní žiadnu funkcionalitu.

6.3 Simulácia siete

Implementácia prototypu pre simuláciu v siete vychádza z návrhu riešenia. Na zmenu parametrov tunelov sa používa nástroj NetEm, ktorý sa ovláda programom TC. Pre aktiváciu, deaktiváciu, zobrazenie a úpravu NetEm profilu sa používa skript v jazyku bash. Počet a čísla portov, ktoré sa použijú pre identifikáciu tunelov a sieťové rozhranie, na ktorom sa profil vytvorí nie je možné zmeniť pomocou vstupných argumentov. Tieto zmeny sa vykonávajú zmenou hodnôt premenných v úvodnej časti skriptu. Spustením skriptu s prepínačom „-h“ sa vypíše požadovaný tvar vstupných argumentov pre vykonanie operácií.

Pre ďalšie zvýšenie prehľadnosti a uľahčenie používania je vytvorené grafické používateľské rozhranie, ktoré môžeme vidieť na obrázku 6.1. Rozhranie je dostupné z webového prehliadača a vnútorne používa vyššie spomenutý skript, ktorý sa spúšťa pomocou PHP funkcie exec. Na vizualizáciu siete sa používa súbor nástrojov(toolkit) JavaScript InfoVis, ktorý sa používa pre interaktívnu vizualizáciu dát na webe. Implementácia rozhrania vychádza z ukážky dostupnej na stránke nástroja. V strednej časti rozhrania sa nachádza graf siete, ktorého veľkosť možno meniť otočením kolieska myši. Umiestnenie uzlov siete môže používateľ meniť ich ťahaním. Po kliknutí na uzol sa

tento zvýrazní a zvýraznia sa aj jeho prepojenia s inými uzlami. Zároveň sa v pravej časti zobrazia rozširujúce informácie, ako napríklad názov uzla a zoznam priamo pripojených uzlov. Pre uzly, ktoré reprezentujú sieťové tunely, sa zobrazia aj parametre tunelov. Po zobrazení parametrov možno ich hodnoty zmeniť a zmenu vykonať stlačením tlačidla v dolnej časti formuláru. Stlačením tlačidla sa všetky potrebné údaje odošlú na server, kde sa po spracovaní použijú ako vstupný argument skriptu. V ľavej časti rozhrania sa nachádza stručný popis rozhrania a tlačidlo, ktoré aktivuje alebo deaktivuje simulátor.



Obrázok 6-1 Grafické používateľské rozhranie nástroja NetEm

6.4 Manažment API

V rámci prototypu manažment serveru sa podarilo vytvoriť a sfunkčniť Node.js aplikáciu, ktorá sa podľa návrhu má starať o dostupnosť aplikačného rozhrania pre všetky časti systému, ktoré to vyžadujú. Prototyp momentálne neobsahuje všetky správy, definované v časti špecifikácia, no rozhodli sme sa implementovať len tie, ktoré potrebujeme pre simuláciu merania kvality na linkách a prenos týchto informácií k zobrazovacej/rozhodovacej časti.

Prototyp sa podľa návrhu spolieha na JavaScript knižnicu Socket.io, ktorá sa stará o komunikáciu jednotlivých častí.

V rámci správ sú implementované tieto:

- HelloServer(id,lines)

  Obsahom tejto správy je jedinečný identifikátor SIRUP - servera a objekt, ktorý popisuje stav pripojených liniek (ich počet, identifikátor linky, meno linky, IP adresy a porty koncových uzlov, maximálna šírka pásma).

- HelloHandler(id)

  Obsahom správy je jedinečný identifikátor servera.
- QualitySet(lines)

  Správa obsahujúca informácie o kvalite liniek prípojných k SIRUP - serveru (celková kvalita linky, packet loss, round trip time, jitter a bandwidth).
- LinesSet(lines)

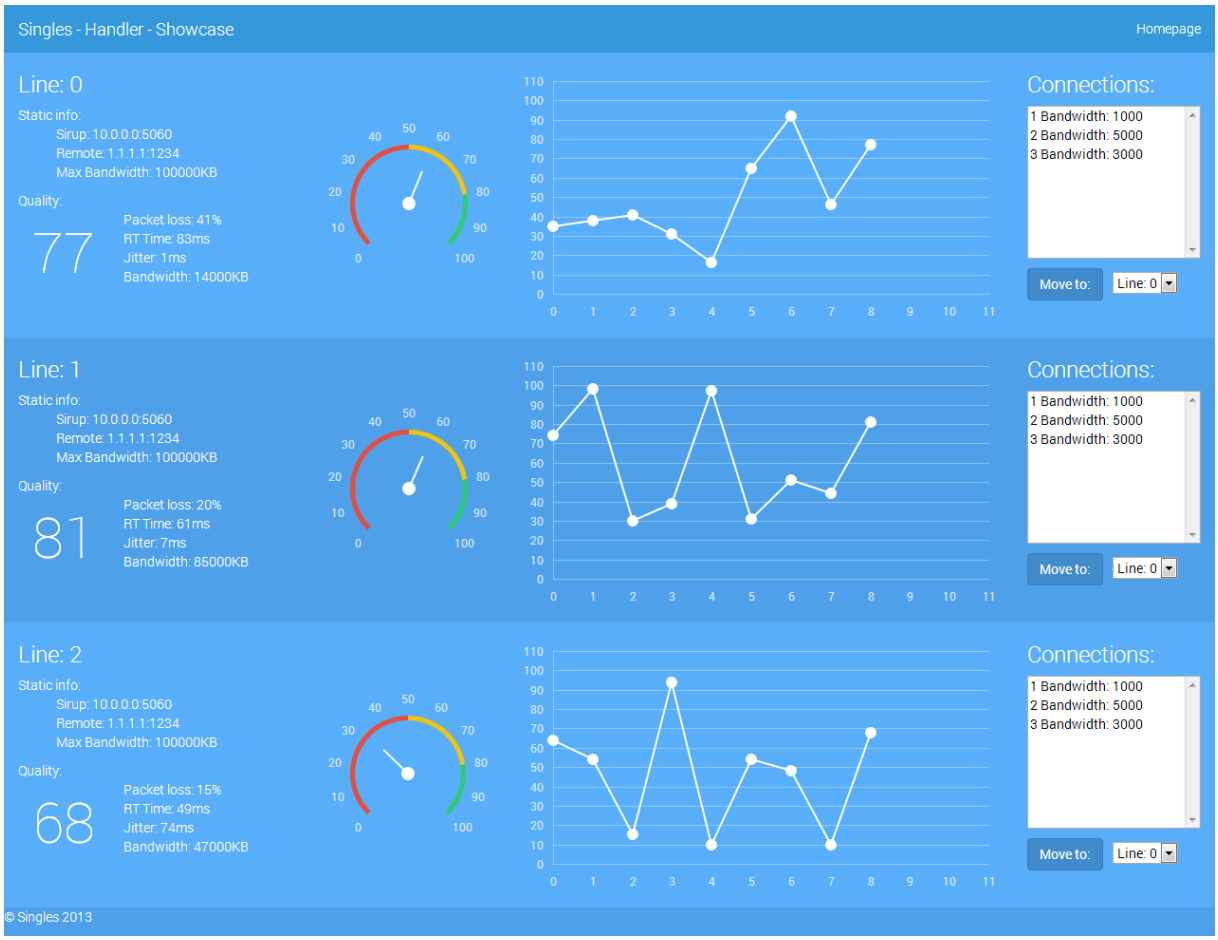
  Správa, ktorá nebola uvedená v špecifikácií, no ukázalo sa, že je vhodná. Manažment server ju posiela zobrazovacej/rozhodovacej jednotke vždy po jej prihlásení (HelloHandler), alebo keď sa zmení stav liniek u SIRUP - servera.

Zobrazovacia/rozhodovacia jednotka

Ďalej bol vytvorený prototyp zobrazovacej/rozhodovacej jednotky. Tak ako bolo špecifikované v návrhu špecifikácií, prototyp má formu webovej aplikácie. Táto aplikácia je schopná prihlásenia sa k manažment serveru a zobrazovania informácií o linkách a kvalite na týchto linkách.

Informácie sú zobrazované v dvojakej forme a to pomocou grafov a textu. V rámci návrhu sme sa rozhodli na zobrazovanie použiť knižnicu ChartJS a tá je použitá aj v prototype. Používajú sa 2 typy grafov a to tzv. „gauge“ (budík na palubnej doske), ktorý prehľadne zobrazuje informáciu o celkovej kvalite linky. Druhý je klasický dvojosový graf, kde sa zobrazuje okrem okamžitej celkovej kvality, aj niekoľko informácií historicky zaznamenaných. Ďalšie informácie o kvalite a statické informácie o linkách sa zobrazujú vo forme textu.

Nefunkčný náhľad ponúka časť „connections“, ktorá bude zobrazovať spojenia na jednotlivých linkách so šírkou pásma, ktoré zaberajú. V rámci tejto časti sa bude ovládať presúvanie spojení na jednotlivé linky.



Obrázok 6-2 Zobrazovacia/rozhodovacia jednotka

V rámci prototypu nie je vytvorené prepojenie medzi SIRUP - serverom a manažment serverom. Aby bolo možné simulovanie zasielania informácií o linkách a ich kvalite, bola vytvorená webová aplikácia, ktorá supluje túto časť funkcionality SIRUP - serveru. Webová aplikácie zasielané dáta pseudonáhodne vytvára a vo finálnej implementácií nebude použitá.

7 Literatúra

[1] About JavaScript - JavaScript | MDN. [Online] https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.

[2] *node.js.* [Online] http://nodejs.org/.

[3] *Chrome V8 — Google Developers.* [Online] https://developers.google.com/v8/.

[4] *Callback Hell.* [Online] http://callbackhell.com/.

[5] *npm.* [Online] https://npmjs.org/.

[6] *Building Web APIs with Node.js and MongoDB.* [Online] http://www.codemag.com/Article/1210041.

[7] *DailyJS: The State of Node and Relational Databases.* [Online] http://dailyjs.com/2013/04/15/node-database-library/.

[8] *MongoDB.* [Online] http://www.mongodb.org/.

[9] *Apache CouchDB.* [Online] http://couchdb.apache.org/.

[10] *JSON.* [Online] http://www.json.org/.

[11] *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST).* [Online] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[12] *Socket.IO: the cross-browser WebSocket for realtime apps.* [Online] http://socket.io/.

[13] ChartJS Dev Express - JavaScript Tool 14.11.2013, [Online] www.chartjs.devexpress.com

[14] Comparison of JavaScript charting frameworks 14.11.2013 [Online] http://en.wikipedia.org/wiki/Comparison_of_JavaScript_charting_frameworks

[15] Linux Foundation - netem 19. 11. 2009 [Online] www.linuxfoundation.org/collaborate/workgroups/networking/netem

[16] RFC 3261, SIP: Session Initiation Protocol [Online] http://www.ietf.org/rfc/rfc3261.txt

[17] SIP Demystified - Gonzalo Camarillo, 2002