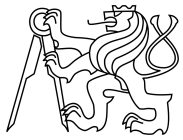
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Kontrola přenosu obrazového signálu
Student:	Jozef Beneš
Vedoucí:	RNDr. Ing. Vladimír Smotlacha, Ph.D.
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Navrhněte a naprogramujte v Linuxu aplikaci pro sledování a vyhodnocení přenosu vysokorychlostního obrazového signálu na úrovni protokolu TCP i UDP pro datové linky 1 Gbps. Pro zachycení vybraných datových toků použijte knihovnu PCAP, pro každý paket uložte parametry zdrojová a cílová adresa, protokol, délka a timestamp. Zpracujte veličiny kapacita datového toku, velikost paketu, časová prodleva mezi pakety a jitter. Hodnoty graficky prezentujte s využitím systému RRDtool. Identifikujte v reálném čase anomálie datového toku a zahlcení datového kanálu.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 11. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalárska práca

Kontrola přenosu obrazového signálu

Jozef Beneš

Vedúci práce: RNDr. Ing. Vladimír Smotlacha, Ph.D.

16. mája 2016

Poďakovanie

Chcel by som sa poďakovať RNDr. Ing. Vladimírovi Smotlachovi, Ph.D. za vedenie bakalárskej práce a zapožičanie techniky potrebnej k jej otestovaniu a Jiřímu Pospíšilíkovi za sprístupnenie gigabitovej linky potrebnej na testovanie tejto práce. Ďalej by som sa chcel poďakovať svojej rodine a priateľom, ktorí ma počas štúdia podporovali.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať.

Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela, a za akýmkoľvek účelom (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené. Každá osoba, ktorá využije vyššie uvedenú licenciu, sa však zaväzuje priradiť každému dielu, ktoré vznikne (čo i len čiastočne) na základe Diela, úpravou Diela, spojením Diela s iným dielom, zaradením Diela do diela súborného či zpracovaním Diela (vrátane prekladu), licenciu aspoň vo vyššie uvedenom rozsahu a zároveň sa zaväzuje sprístupniť zdrojový kód takého diela aspoň zrovnateľným spôsobom a v zrovnateľnom rozsahu ako je zprístupnený zdrojový kód Diela.

V Prahe 16. mája 2016 .....

Odkaz na túto prácu

Beneš, Jozef. *Kontrola přenosu obrazového signálu.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Vo svojej bakalárskej práci sa venujem problematike monitorovania vysokorýchlostne streamovaného videa, detekovaniu chýb a tvorbe štatistík o tomto prenose. Súčasťou práce je analýza protokolov používaných na prenos videa, ako aj predošlých riešení tohto problému. Výstupom tejto bakalárskej práce je aplikácia napísaná v jazyku C pod operačným systémom Linux, ktorá rieši vyššie uvedený problém. Pri implementácii tejto aplikácie bola použitá knižnica pcap, ktorá slúži na odchytávanie a analýzu paketov. Na uchovávanie štatistík a tvorbu grafov bol použitý nástroj RRDtool. Uvedenú aplikáciu je možné v praxi použiť na prípadnú detekciu slabých miest v sieti pri prenose videa a pomoc pri ich možnej následnej eliminácii. Na priloženom CD je možné nájsť zdrojové kódy výslednej aplikácie a výsledky testov jej funkčnosti.

Klíčová slova aplikácia, monitorovanie, prenos videa, jazyk C/C++, pcap, RRDtool

Abstract

In my bachelor thesis I deal with the issue of monitoring of high-speed streaming video, detecting errors and providing statistics about this stream. Part of the thesis is the analysis of protocols used for video streaming, as well as previous solutions of this problem. The outcome of this thesis is an application written in C under the Linux operating system, which solves the above problem. For implementation of this application was used library pcap, which serves for the packet capture and analysis. For keeping statistics and as charting tool was used RRDtool. This application can be practiced to detect possible weak points in the network for video streaming and help in their possible subsequent elimination. On the enclosed CD you can find the source code of the resulting application and test results of its functionality.

Keywords application, monitoring, video streaming, language C/C++, pcap, RRDtool

Obsah

Zoznam obrázkov

Zoznam ukážok kódu

Úvod

V súčasnosti sa stále zvyšuje trend lepšieho rozlíšenia obrazu, či už pri rozlíšení obrazoviek alebo rozlíšení prenášaného videa. Rovnako sa zväčšuje využitie internetu vo všetkých oblastiach života, od práčiek až po automobily. Postupne teda vznikla potreba prenášať video vo vysokom rozlíšení cez internet, z čoho vyplynuli aj problémy, ktoré je potrebné riešiť. Jedným z nich je požiadavka monitorovať tento prenos a chyby v ňom. Z vyššie uvedených dôvodov som si vybral túto tému.

V práci sa zaoberám implementáciou nástroja, ktorý je schopný v reálnom čase ukladať vybrané informácie o jednotlivých paketoch do súboru, vytvárať štatistiky o prenášanom videu, graficky ich znázorňovať a v prípade chýb v prenose videa upozoniť užívateľa na tieto anomálie.

Kapitola 1

Cieľ práce

1.1 Cieľ rešeršnej časti práce

Cieľom rešeršnej časti práce je získať prehľad o technológiách používaných pri prenose obrazového signálu cez internet. Medzi tieto technológie patria najmä protokoly slúžiace na prenos videa, ktoré sú v práci obsiahnuté. Ďalej sa venujem už existujúcim riešeniam, ktoré sa riešia danú problematiku a popisu knižníc, ktoré pri implementácii praktickej časti práce používam.

1.2 Cieľ praktickej časti práce

Cieľom praktickej časti práce je vytvoriť nástroj, ktorý bude primárne slúžiť na kontrolu prenosu videa cez počítačovú sieť. O každom pakete bude zaznamenávať základné údaje ako zdrojová a cieľová IP adresa, protokol, dĺžku paketu a jeho časovú známku. Ďalej bude spracúvať veličiny ako jitter, kapacita dátového toku, veľkosť paketu a časová medzera medzi paketmi, ktoré bude ukladať do round robin databázy. Tieto údaje budú použité na generovanie grafov. Aplikácia je napísaná pod systémom Linux a používa knižnice libpcap a librrd (RRDtool), ktorých popis je súčasťou práce. Ďalej bude tento nástroj užívateľa upozorňovať na anomálie v dátovom toku ako aj na jeho zahltenie.

Kapitola 2

Úvod do problematiky

Video sa rýchlo po svojom vzniku stalo významným prvkom kultúry. Spočiatku sa jeho prenos a zaznamenávanie uskutočňovalo v analógovej podobe, no s postupom času a príchodom počítačov sa začalo s jeho digitalizáciou. Ďalší vývoj nastal v 90. rokoch 20. storočia, kedy sa začal vo väčšej miere rozširovať internet a hľadali sa spôsoby ako vysielať video aj prostredníctvom tohto média.

2.1 Prenos videa cez internet

Kapitola čerpá zo zdrojov [1, 2, 3].

Prvým významnejším spôsobom, ako bolo video streamované bolo streamovanie pomocou datagramov, teda použitím protokolov využívajúcich hlavne protokol UDP.

Začali vznikať proprietárne protokoly založené na tomto princípe, ako napr. Microsoft Media Server (MMS) od Microsoftu alebo Real Time Messaging Protocol (RTMP) od spoločnnosti Adobe, ktoré boli ale postupne vytlačené neproprietárnym protokolom Real-time Transport Protocol (RTP). Ten sa vo veľkej miere používa aj dodnes, napr. pri VoIP. Na svoje fungovanie potrebuje signálovanie, na čo mu slúžia protokoly Real Time Streaming Protocol (RTSP) alebo Session Initiation Protocol (SIP).

Má aj svoje nedostatky, ako je jeho závislosť iba na ním štandardizovaných kodekoch, problematický prechod cez firewall a nutnosť kontrolovať prijaté pakety, či sú v správnom poradí, či niektoré nechýbajú atď.

Tieto nedostatky sa podarilo vyriešiť pomocou progresívneho sťahovania videa použitím HTTP. Ide vlastne o sťahovanie súboru obsahujúceho video,

ktoré je možné prehrávať počas sťahovania. Táto metóda je ešte v súčasnosti využívaná najmä v spojení s platformou Adobe Flash alebo Microsoft Silverlight. Pri tomto spôsobe však vyvstali problémy s väčšou veľkosťou bufferu a častejším prerušovaním videa, čo komplikuje až znemožňuje jeho použitie v komunikácii v reálnom čase.

Poslednou, v súčasnosti najmodernejšou metódou je adaptívne streamovanie s použitím HTTP. Pri tomto najnovšom prístupe sa súbor obsahujúci video rozdelí na množstvo menších segmentov, dlhých približne 2 až 4 sekundy. Tieto časti, tzv. chunky sú uložené na HTTP serveri a odtiaľ postupne vyžadované klientskou aplikáciou. Chunky sa potom sťahujú metódou progresívneho sťahovania. Na serveri je uložené video vo viacerých bitratoch súčasne.

Prerušovania videa rieši výberom videa s nižším bitratom, čím ho činí vhodným pre real-timové použitie. Funguje tak, že na serveri je dostupné video v rôznych typoch kvality a na klientovi sa zvolí na základe jeho rýchlosti pripojenia prislúchajúci bitrate.

V súčasnosti dostupné formáty tohto typu streamovania videa sú napr. HTTP Live Streaming (HLS) od Applu, HTTP Dynamic Streaming (HDS) od Adobe, Smooth Streaming od Microsoftu a ISO štandard MPEG-DASH. Tento spôsob prenosu videa v súčasnosti požívajú aj najväčšie spoločnosti zaoberajúce sa streamovaním videa YouTube a Netflix.

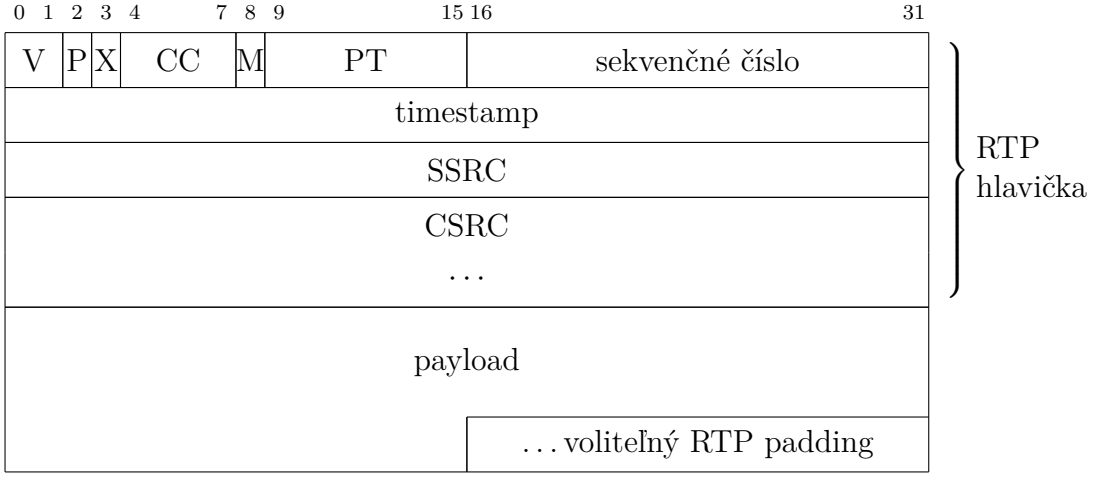
Oblasť prenášania videa cez internet je vskutku perspektívna, čo ukázal odhad spoločnosti Cisco, ktorý tvrdí, že do roku 2019 vzrastie podiel videa na konzumentskom internetovom trafficu na 80 % zo 64 %, ktorý dosiahol v roku 2014 [4].

2.2 Protokoly

Kapitola čerpá zo zdrojov [5, 6, 7].

2.2.1 RTP

Prvý protokol slúžiaci na prenos videa cez internet, ktorému sa budem venovať je Real-time Transport Protocol, skrátene RTP. Tento protokol aplikačnej vrstvy bol vyvinutý skupinou Audio-Video Transport Working Group, ktorá je súčasťou Internet Engineering Task Force a prvýkrát publikovaný ako RFC 1889 v januári 1996. V júli 2003 bol aktualizovaný a dnes je popísaný v RFC 3550.



Obr. 2.1: Štruktúra paketu protokolu RTP

Tieto štandardy okrem RTP upravujú aj Real-time Transport Control Protocol, skrátene RTCP, ktorý sa v spojení s RTP používa primárne na získavanie údajov o kvalite služby (QoS).

Významy jednotlivých polí hlavičky RTP protokolu sú takéto:

- **V**: Verzia protokolu (2 bity), v súčasnosti hodnota 2.
- **P**: Nastavenie paddingu (1 bit).
- **X**: Indikátor prítomnosti rozširujúcich hlavičiek medzi hlavičkou a dátami (1 bit).
- **CC**: Počet CSRC identifikátorov (4 bity).
- **M**: Marker (1 bit), jeho interpretácia je závislá na profile.
- **PT**: Typ payloadu (7 bitov), východzie mapovanie je určené v RFC 3551.
- **sekvenčné číslo**: Číslo (16 bitov), ktoré sa zvyšuje každým odoslaným RTP paketom. Slúži prijímateľovi na zistenie straty paketov, resp. ich nesprávneho poradia.
- **timestamp**: Časová známka (32 bitov), slúži prijímateľovi na určenie správnych intervalov, po ktorých sa majú pakety prehrať. Na začiatku je inicializovaná náhodnou hodnotou.
- **SSRC**: Identifikátor zdroja streamu (32 bitov).

- **CSRC**: Identifikátory (32 bitov každý) prispievajúcich zdrojov (0 až 15).

Ako naznačuje položka hlavičky *typ payloadu*, protokol RTP je obmedzený v používaní len tých kodekov, ktoré už majú priradené číslo, t.j. sú štandardizované. Rozširovanie tejto množiny kodekov je pomerne nepružné pre nutnosť novelizovať RFC.

Ďalšou vlastnosťou protokolu RTP je potreba kontrolovať poradie paketov a aj prípadné chýbajúce pakety na aplikačnej vrstve kvôli prevažujúcemu používaniu nepotvrdzovaného protokolu UDP na transportnej vrstve.

Na vytvorenie a spravovanie spojenia pomocou protokolu RTP je v praxi často nutné využívať signalizačné protokoly ako H.323, Session Initiation Protocol (SIP) alebo Jingle (XMPP). Najpoužívanejší z nich je protokol SIP založený na textovej forme s transakčným modelom podobným HTTP.

Protokol RTP taktiež ponúka možnosť šifrovať payload, na čo má určený špeciálny profil Secure Real-time Transport Protocol (SRTP), ktorý je upravený v RFC 3711 vydanom v marci 2004. Na šifrovanie vo východzom nastavení sa používa bloková šifra AES v jednom z dvoch módov, ktoré jej umožňujú, aby bola použitá ako prúdová šifra.

2.2.2 MPEG-DASH

MPEG-DASH (Dynamic Adaptive Streaming over HTTP) je v súčasnosti jeden z najnovších spôsobov prenosu videa. Tento prenos je realizovaný použitím adaptívneho streamovania cez HTTP, ako napovedá jeho názov.

Vznikol na podnet skupiny MPEG v reakcii množstvo nekompatibilných riešení v tejto oblasti (HLS, HDS, Smooth Streaming, atď.) a s tým súvisiaci chýbajúci štandard. Na štandarde sa začalo pracovať na prelome rokov 2009 a 2010 a finálny štadard bol publikovaný v apríli 2012 ako ISO/IEC 23009-1:2012.

Tento štandard podporuje DRM, je nezávislý na kodekoch a požaduje len použitie dvoch kontajnerov, ISO BMFF (v podstate MP4) alebo MPEG-2 TS, ktoré už boli používané v HDS a Smooth Streamingu, resp. HLS. Vďaka tomuto pri prechode na DASH nie je nutné meniť chunky súborov na serveri, len indexové súbory, ktoré ich popisujú (umiestnenie, bitrate a pod.), musia byť prevedené do formátu MPD (Media Presentation Description).

Kapitola 3

Existujúce riešenia

V tejto kapitole sa venujem súčasnému stavu v oblasti monitorovania prenosu videa. Prvé dva nástroje nie sú nijako špecializované na monitorovanie videa, posledné tri sú plne dedikované monitorovaniu prenosu videa, alebo je táto funkcionalita ich súčasťou.

3.1 tcpdump

Tcpdump je robustný nástroj slúžiaci na odchytávanie paketov a výpis informácií o nich. Na jeho vytvorení sa podieľali Van Jacobson, Craig Leres a Steven McCanne z Lawrence Berkeley National Laboratory na University of California [8].

Tcpdump síce nemá grafické užívateľské rozhranie, ale ponúka širokú paletu filtrov, ktoré je možné vo vhodnej kombinácii nastavení použiť aj na monitorovanie prenosu videa. Je dostupný pre unixové systémy, ale existujú aj podobné alternatívy pre systém Windows, napr. WinDump. Svoju funkcionalitu opiera o knižnicu libpcap, ktorú vo svojej práci používam aj ja.

3.2 Wireshark

Wireshark (pôvodne Ethereal) je až na pár funkcií navyše len grafická nadstavba nástroja tcpdump. Na jeho vývoji sa podieľal Gerald Combs, ktorý vydal jeho prvú verziu v roku 1998 [9].

Medzi spomínané funkcie navyše patrí možnosť odchytávať komunikáciu USB, prehrávanie odchyteného VoIP telefonátu alebo možnosť rozšíriť pluginmi o schopnosť analyzovať ďalšie protokoly.

3.3 Dotcom-Monitor ServerView

Platený nástroj od americkej spoločnosti Dotcom-Monitor[10] využíva na správu monitorovania webové rozhranie. To umožňuje zadať monitorovacie úlohy[11], ktoré spočívajú v tom, že agent sa pripojí na zadaný stream, načíta a prehrá 3 až 5 sekúnd videa a potom sa odpojí.

Ak pri vykonávaní úlohy je stream nedostupný alebo monitorované metriky (framerate, počet bytov za sekundu, trvanie pripájania, trvanie načítania) vykazujú anomálie, pošle užívateľovi upozornenie vo forme e-mailu, SMS alebo na pager.

Monitorovacie agenty majú rozmiestnené po celom svete, čím sa dá aj overiť celosvetová dostupnosť streamu. Podporujú množstvo protokolov, ktoré sa na streamovanie videa používajú (medzi nimi aj RTP aj HTTP).

3.4 SciVisum Media Streaming Monitor

Ďalší nástroj je tiež platený, a je od britskej spoločnosti SciVisum[12]. Sleduje rovnaké metriky ako predošlý nástroj, upozornenia posiela e-mailom alebo cez SMS, tiež má monitorovacie agenty distribuované po celom svete. Má o dosť menšiu podporu protokolov, a to len HTTP, RTMPE a RTMPS (posledné 2 sú proprietárne protokoly spoločnosti Adobe).

Výhodou oproti predošlému nástroju je ale možnosť zobrazenia výsledkov monitorovania a historický vývoj meraní v grafoch.

3.5 Radcom MaveriQ Service Assurance

Posledný komerčný nástroj je od izraelskej firmy Radcom [13]. Zameriava sa hlavne na sledovanie kvality vnímania (Quality of Experience – QoE). V reálnom čase zobrazuje kľúčové výkonnostné ukazatele (Key Performance Indicators – KPI) ako chybovosť, čas od pripojenia k odštartovaniu streamu, kvalitu audia a videa príčinu ukončenia streamu a podobné.

V prípade, že nástroj zdeteguje anomálie v sledovaných hodnotách, poskytuje možnosť proaktívneho alarmu, ktorý na ne upozorní.

Kapitola 4

Analýza

V tejto kapitole sa zaoberám analýzou mnou vytvorenej aplikácie. Postupne rozoberám funkčné a nefunkčné požiadavky, ktoré by mala výsledná aplikácia spĺňať. Ďalej predstavujem prípady použitia tejto aplikácie a nakoniec kapitolu uzatváram analýzou použitých technológií.

4.1 Funkčné požiadavky

Na aplikáciu sú kladené tieto funkčné požiadavky:

- Pre vybrané dátové toky má ukladať o každom pakete zdrojovú a cieľovú IP adresu, protokol transportnej vrstvy, jeho dĺžku a časovú známku.
- Je vyžadovaná podpora transportných protokolov TCP a UDP.
- Aplikácia musí zvládnuť spracovať pakety na linkách s rýchlosťou do 1 Gbps.
- Ďalej má aplikácia o zvolenom dátovom toku spracovať kapacitu dátového toku, veľkosť paketu, časovú medzeru medzi paketmi a jitter. Vypočítané hodnoty má aplikácia ukladať a ich vývoj v čase následne graficky prezentovať.
- Užívateľ má byť aplikáciou upozornený na prípadné anomálie v dátovom toku alebo na jeho zahltenie.

4.2 Nefunkčné požiadavky

Zo zadania a konzultácií vyplynuli takéto nefunkčné požiadavky:

- Aplikácia má byť vyvinutá nad operačným systémom z rodiny UNIX.
- Jednotlivé pakety majú byť odchytávané pomocou knižnice PCAP (libpcap).
- Vybrané hodnoty, ktoré aplikácia vypočíta, majú byť ukladané a následne zobrazené použitím nástroja RRDtool.
- Výsledné grafy majú byť zobrazené pomocou webovej stránky.

4.3 Prípady použitia

4.3.1 Odchytávanie paketov

1. Užívateľ pomocou prepínačov skriptu určí sledované spojenie a zvolí odchytávanie paketov.
2. Aplikácia zvaliduje užívateľom zadané parametre.
3. Následne spustí odchytávanie paketov a detekciu chýb v spojení.
4. Z odchytených paketov sa generuje CSV súbor, z ktorého aplikácia dokáže v ďalšom behu vypočítať žiadané hodnoty.

4.3.2 Spracovanie paketov

1. Užívateľ pomocou prepínačov skriptu zvolí spracovanie paketov a určí vstupný CSV súbor.
2. Aplikácia zvaliduje užívateľom zadané parametre.
3. Následne sa zaháji spracovanie paketov a plnenie round robin databázy.
4. Z round robin databázy je potom každú sekundu generovaný graf pre každú zo sledovaných hodnôt. Grafy sú dostupné z webovej stránky.

4.3.3 Odchytávanie a spracovanie paketov

V tomto prípade aplikácia postupne spracuje výstup odchytávania, teda výkonáva funkcionalitu popísanú v 4.3.1 a 4.3.2 a generuje pri tom grafy vývoja vypočítaných hodnôt.

4.4 Použité technológie

4.4.1 Jazyk C++

Táto časť čerpá z [14].

Jazyk C++ je objektovo orientovaný, imperatívny programovací jazyk. Na základe jazyka C ho vyvinul Bjarne Stroustrup. S jeho vývojom sa začalo v roku 1979 a prvá verzia bola použitá interne v AT&T v auguste 1983. Jazyk C++ je štandardizovaný ISO. Jeho prvý štandard bol publikovaný v roku 1998 a v súčasnosti je v platnosti najnovšia verza štandardu z roku 2014.

Keďže je jazyk C++ založený na jazyku C, zachoval si aj kompatibilitu s jeho knižnicami a taktiež napríklad jeho schopnosť pracovať s pamäťou na nízkej úrovni. Jazyk C++ priniesol so sebou aj knižnicu Standard Template Library (STL), ktorá obsahuje okrem iného generické kontajnery. Tieto kontajnery si riešia správu pamäte sami, čím znižujú riziko memory leakov, na ktoré sú jazyky C a C++ náchylné, keďže správa pamäte je inak v kompetencii programátora.

Pre implementáciu som si jazyk C++ vybral najmä kvôli podpore knižníc libpcap a librrd (knižnica na prácu s RRDtool), ktoré som pri vývoji použil, knižnici STL a v neposledom rade aj kvôli jeho rýchlosti oproti iným programovacím jazykom.

Jeho rýchlosť okrem iného skúmal v roku 2011 Robert Hundt zo spoločnosti Google v štúdii, ktorá porovnávala výkon jazykov C++, Java, Scala a Go pri implementácii Tarjanovho algoritmu. Vo výsledku spracovanie testovacej sady trvalo programu napísanému v C++ 23 sekúnd a najviac sa mu z ostatných jazykov priblížila Scala s časom 58 sekúnd, čo je zhruba 2,5-krát dlhší čas [15].

4.4.2 libpcap

Táto časť čerpá z [8, 16].

Libpcap je open-source knižnica, ktorá slúži na odchytávanie a filtráciu paketov prechádzajúcich sieťovým zariadením. Za jej vytvorení v roku 1994 stoja výskumníci Van Jacobson, Craig Leres a Steven McCanne z Lawrence Berkeley National Laboratory na University of California.

Libpcap je vytvorená a prispôsobená na priame použitie v programovacích jazykoch C a C++, ale existujú aj rôzne wrappery, ktoré umožňujú jej požitie v iných jazykoch ako napríklad Python (napr. modul scapy) alebo Java (JNetPcap). Je taktiež požiteľná na väčšine UNIX-ových systémov (Linux, Solaris, BSD, ...) a existuje tiež wrapper pre jej použitie na systémoch Windows (WinPcap). V súčasnosti libpcap vyvíja a udržiava Tcpdump Group a na ich webovej stránke je možné nájsť aj dokumentáciu k tejto knižnici.

Libpcap na filtrovanie paketov používa BSD Packet Filter (BPF), ktorý bol vyvinutý okolo roku 1991 rovnakými výskumníkmi ako libpcap. Pri svojom predstavení v roku 1993 bol 10 až 150-krát rýchlejší ako NIT od spoločnosti Sun a 1,5 až 20-krát rýchlejší ako filter CSPF pri približne rovnakom zaťažení a na rovnakom hardvéri.

BPF poskytuje filtrovanie paketov na úrovni jadra operačného systému, a to pomocou programov napísaných v jazyku pripomínajúcom assembler. Poskytuje však aj jazyk na zápis filtra, ktorý je užívateľsky prívetivejší a kompiluje sa na výsledný BPF program. Každý operačný systém si implementuje vlastný paketový filter, ale mnohé z nich (napr. BSD, Linux a MacOS) sú založené na BPF a teda je s nimi libpcap kompatibilný.

Knižnicu libpcap som si vybral hlavne preto, že je open-source a nie som si vedomý toho, že by k nej existovali nejaké ekvivalentné náhrady. Ďalším dôvodom je aj to, že libpcap je medzi nefunkčnými požiadavkami tejto bakalárskej práce.

4.4.3 Round robin databáza a RRDtool

Táto časť čerpá z [17, 18].

Round robin databáza je databáza s vopred určeným počtom miest na dáta. Do databázy je teda uložený konštantný počet posledných nameraných hodnôt. Vďaka tejto vlastnosti veľkosť databázy nikdy neprevýši určenú veľkosť. S takýmito databázami pracuje RRDtool.

RRDtool je open-source nástroj slúžiaci na prácu s round robin databázami a tvorbu grafov z dát v nich uložených. Vyvinul ho švajčiarsky vývojár

Tobias Oetiker. Pri vytváraní round robin databázy pomocou RRDtoolu sa zadávajú zdroje dát (Data Source – DS) a round robin archívy (RRA).

Zdroju dát je potrebné zadať meno, typ (GAUGE, COUNTER, DERIVE, ABSOLUTE), heartbeat, minimálnu a maximálnu hodnotu. Pri type GAUGE sa do databázy uloží zadaná hodnota, pri type COUNTER sa do databázy uloží minulá hodnota inkremetovaná o zadanú hodnotu a pri type DERIVE sa do databázy uloží derivácia priamky vedúcej z minulého do súčasného dátového bodu. Heartbeat označuje maximálnu dobu medzi 2 updatami databázy, po ktorej bude do databázy uložená hodnota UNKNOWN (neznáma hodnota).

Pri round robin archívoch sa zadáva konsolidačná funkcia, xfiles faktor (XFF), počet krokov a počet riadkov. Počet riakdov určuje, koľko hodnôt sa celkovo uloží v danom RRA. Počet krokov určí, z koľkých vložených hodnôt sa bude konsolidovať pomocou zadanej funkcie na jednu hodnotu, ktorá sa nakoniec do RRA uloží. Parameter XFF určuje, aký podiel z hodnôt, ktoré sa budú konsolidovať, može byť typu UNKNOWN na to, aby výsledná konsolidovaná hodnota nebola UNKNOWN. Konsolidačná funkcia určuje, ako sa spracujú dotknuté body do RRA. Funkcia AVERAGE zo zadaných hodnôt vloží do RRA priemer, funkcie MAX a MIN vložia do RRA najväčšiu, resp. najmenšiu hodnotu a funkcia LAST do RRA vloží poslednú, najnovšiu hodnotu.

RRDtool je tým, že používa round robin databázu, ktorá nikdy neprerastie určitú veľkosť, a tým, že umožňuje dáta rozumne agregovať pomocou konsolidačných funkcií, ideálny pre použitie v aplikáciách, ktoré zaznamenávajú nejaké údaje v priebehu času. Takou je aj moja aplikácia vyvinutá vrámci tejto bakalárskej práce.

4.4.4 JavaScript

JavaScript je skriptovací programovací jazyk používaný najmä na vývoj dynamických webových stránok. Vyvinul ho Brendan Eich vtedy pracujúci pre spoločnosť Netscape za 10 dní v máji 1995 [19].

JavaScript som si zvolil pre implementáciu grafových výsledkov svojej bakalárskej práce hlavne kvôli tomu, že na rozdiel od jazyka PHP, ktorý sa tiež využíva pri dynamických webových stránkach, nevyžaduje webový server, na ktorom by bežal, ale beží len vo webovom prehliadači.

4.4.5 Python

Python je skriptovací programovací jazyk so širokými možnosťami využitia. Za jeho vznikom stojí Guido van Rossum, ktorý s jeho vývojom začal

na konci 80. rokov minulého storočia [20]. V súčasnosti sú používané verzie 2 a 3, ktoré medzi sebou nie sú kompatibilné.

V bakalárskej práci som sa rozhodol pre použitie Pythonu 3 pre implemetáciu spúšťacieho skriptu najmä kvôli modulu argparse, ktorý poskytuje mocný nástroj na parsovanie argumentov zadaných skriptu z príkazového riadku a modulu subprocess, ktorý poskytuje funkcie na spúšťanie ďalších procesov a riadenie komunikácie medzi nimi.

Kapitola 5

Implementácia

V tejto kapitole sa venujem popisu jednotlivých súčastí a technikám, ktoré som na dosiahnutie požadovaných výsledkov použil.

5.1 Spúšťací skript

Hlavnou úlohou skriptu, ktorý implementovanú aplikáciu spúšťa, je poskytnúť užívatelovi príjemnejšie rozhranie na prácu s ňou. Ďalej zaisťuje správne prepojenie jednotlivých súčastí aplikácie a taktiež validáciu zadaných parametrov.

Prívetivejšie užívateľské rozhranie skript užívateľovi dáva prostredníctvom prepínačov a parametrov, ktoré je možné skriptu zadať. Sú 2 hlavné prepínače, ktoré ďalej určujú, aká časť aplikácie bude spustená.

- Pri zadaní prepínača `--sniff` sa spustí iba odchytávanie paketov. Informácie o paketoch potrebné pre ich ďalšie spracovanie budú ukladané do zadaného súboru vo formáte CSV.

- Pri zadaní prepínača `--process` bude spracovaný zadaný CSV súbor získaný predošlým spustením skriptu s prepínačom `--sniff`. Pri spracovaní budú generované grafy veličín popisujúcich sieťové spojenie a budú detegované aj prípadné anomálie, ktoré nastali pri získaní súboru.

- Pri spustení skriptu bez prepínačov `--sniff` alebo `--process` prebehne odchytávanie paketov aj ich spracovanie súčasne. To znamená, že obe súčasti aplikácie budú pracovať súčasne prepojené rúrou a generovanie grafov aj detekcia anomálií budú prebiehať v reálnom čase.

5.2 Odchytávanie paketov a ich predspracovanie

Odchytávač paketov má ako súčasť aplikácie 2 parametre, názov sieťového rozhrania, ktoré sa má sledovať (nie je povinný) a reťazec obsahujúci filter paketov. Na začiatku sa musí kvôli filtrovaniu paketov o zadanom, prípadne východzom zariadení zistiť jeho IP adresa a maska a následne je potrebné ho otvoriť, aby bolo možné odchytávať pakety. Kvôli prevencii zahadzovania paketov z dôvodu zaplnenia paketového bufferu som jeho veľkosť zväčšil na 4 MiB.

Potom sa nastaví paketový filter. Knižnica libpcap používa filter BPF a má rovnakú syntax ako program tcpdump. Ak aj užívateľ nezadá v spúšťacom skripte žiadne filtrovacie parametre, v skripte sú určené východzie hodnoty, a to, že sa majú zachytávať len pakety protokolov TCP a UDP s vybranými cieľovými a zdrojovými portmi:

- Porty 80 a 443 kvôli protokolom, ktoré na prenos videa používajú protokol HTTP, resp. HTTPS (napríklad MPEG-DASH).
- Port 554 kvôli protokolu RTSP.
- Ostatné porty s výnimkou well-known portov (rozsah 0 až 1023, ako je uvedené v RFC 6335 [21]) kvôli ostatným protokolom, napr. RTP.

Analýza paketov potom prebieha postupným zisťovaním hlavičiek jednotlivých vrstiev ISO/OSI modelu. Funkcia `pcap_loop()`, ktorá odchytáva a filtruje pakety, volá na každý paket, ktorý filtru vyhovel, funkciu `callback (u_char*, const struct pcap_pkthdr*, const u_char*)`. Signatúra tejto funkcie je fixná a v takejto podobe vyžadovaná knižnicou libpcap. Táto funkcia v argumente `packet` obsahuje paket vo forme bajtov. Ako vidieť v ukážke 5.1, rozkladanie paketu na jednotlivé vrstvy prebieha pomocou pretypovania argumetu `packet` posunutého o predošlé hlavičky na hlavičku zisťovanej vrstvy. Štruktúry, ktoré na pretypovanie používam, sú definované v knižniciach `net/ethernet.h`, `netinet/ip.h`, `netinet/ip6.h`, `netinet/udp.h` a `netinet/tcp.h`. Ďalej som pre použitie konštánt (napr. AF_INET alebo INET_ADDRSTRLEN) a funkcií na prácu s paketmi, internetovými adresami a protokolmi (napr. `ntohs()` alebo `inet_ntop()`) použil knižnice `netdb.h` a `arpa/inet.h`.

Ukážka 5.1: Ukážka funkcie callback()

```
void callback(u_char* args, const struct pcap_pkthdr* pkthdr
    , const u_char* packet) {
    struct ether_header* ethernet = (struct ether_header*) (
       packet);
    u_short etherType = ntohs(ethernet->ether_type);
    protoent *protocol;
    string pktParams;
    if (etherType == ETHERTYPE_IP) {
        char src[INET_ADDRSTRLEN], dst[INET_ADDRSTRLEN];
        struct ip *ip4 = (struct ip*) (packet + sizeof (
           struct ether_header));
        u_int len = ntohs(ip4->ip_len) - ip4->ip_hl * 4;
        protocol = getprotobynumber(ip4->ip_p);
        inet_ntop(AF_INET, &(ip4->ip_src), src,
           INET_ADDRSTRLEN);
        inet_ntop(AF_INET, &(ip4->ip_dst), dst,
           INET_ADDRSTRLEN);
        if (protocol->p_proto == 6) {
            pktParams = processTcp(packet, etherType, len);
        } else if (protocol->p_proto == 17){
            pktParams = processUdpRtp(packet, etherType);
            if (pktParams == "") {
            return;
            }
        } else {
            return;
        }
        cout << src << ";" << dst << ";" << protocol->p_name
            <<  ";" << pkthdr->len << ";" << pkthdr->ts.
           tv_sec << "." << pkthdr->ts.tv_usec << ";" <<
           pktParams << endl;
    } else ... // IPv6 pakety
```

Po zistení, aký protokol bol v pakete použitý na transportnej vrstve, bude spustená buď funkcia `processTcp()` pre protokol TCP alebo funkcia `processUdpRtp()` pre protokol UDP. Tieto funkcie majú za úlohu zistiť z paketov údaje, ktoré budú potrebné na detegovanie anomálií v dátovom toku. Pri protokole TCP to sú zdrojový a cieľový port, sekvenčné a potvrdzovacie číslo paketu a príznaky paketu.

Ukážka 5.2: Funkcia processTcp()

```
string processTcp(const u_char* packet, const u_short&
    etherType, const u_int& len){
     struct tcphdr* tcp;
     if (etherType == ETHERTYPE_IP) {
         tcp = (struct tcphdr*) (packet + sizeof (struct
            ether_header) + sizeof (struct ip));
     } else {
         tcp = (struct tcphdr*) (packet + sizeof (struct
            ether_header) + sizeof (struct ip6_hdr));
     }
     string res = to_string(ntohs(tcp->source)) + ";" +
        to_string(ntohs(tcp->dest)) + ";" + to_string(ntohl(
        tcp->seq)) + ";" + to_string(ntohl(tcp->ack_seq)) + "
        ;" + to_string(len - tcp->th_off * 4) + ";" +
        to_string(tcp->th_flags);
     return res;
}
```

Pri protokole UDP prebieha ešte zisťovanie hlavičky aplikačnej vrstvy paketu. Keďže pri protokole UDP je na rozdiel od protokolu TCP kontrola správne prebehnutého prenosu dát na vyššej vrstve, je potrebné pre každý protokol aplikačnej vrstvy, ktorý sa používa na prenos videa, implementovať detekciu anomálií v prenose dát zvlášť. V tejto bakalárskej práci implementujem detekciu anomálií pri použití protokolu UDP na transportnej vrstve len pri použití RTP na vrstve aplikačnej.

Detekcia samotného protokolu RTP v pakete prebieha postupne overením, či dáta UDP paketu nie sú kratšie ako samotná hlavička protokolu RTP a následným overením, či zachytený paket používa protokol RTP verzie 2. Táto detekcia však nie je úplne neomylná, takže sa môže stať, že ňou prejdú pakety iného protokolu ako RTP.

Keď paket prejde touto kontrolou, funkcia o ňom vráti údaje potrebné na detekciu anomálií, a to v prípade protokolu RTP zdrojový a cieľový port a sekvenčné číslo paketu (časť hlavičky protokolu RTP).

Ukážka 5.3: Funkcia processUdpRtp()

```
string processUdpRtp(const u_char* packet, const u_short&
    etherType){
     struct udphdr* udp;
     struct rtphdr* rtp;
     if (etherType == ETHERTYPE_IP) {
         udp = (struct udphdr*) (packet + sizeof (struct
            ether_header) + sizeof (struct ip));
```

```
        if (ntohs(udp->uh_ulen) - 8 <= sizeof(struct rtphdr)
            ) return "";
        rtp = (struct rtphdr*) (packet + sizeof (struct
            ether_header) + sizeof (struct ip) + sizeof (
            struct udphdr));
        if (rtp->v != 2) return "";
    } else {
        // IPv6 pakety
    }
    string res = to_string(ntohs(udp->source)) + ";" +
        to_string(ntohs(udp->dest)) + ";" + to_string(ntohs(
        rtp->seq_num));
    return res;
}
```

Po zistení údajov potrebných na detekciu anomálií, ktorá prebieha v ďalšej časti aplikácie, sú tieto údaje spolu s údajmi na logovanie zasielané na štanardný výstup oddelené bodkočiarkami, kde tým vytvárajú CSV súbor spracovaný ďalšou časťou aplikácie.

5.3 Spracovanie

Časť aplikácie, ktorej úlohou je spracovať pakety, detegovať anomálie a generovať grafy pracuje v cykle, ktorý načítava zo štandardného vstupu CSV riadky, ktoré si následne parsuje, aby s nimi mohol ďalej pracovať. Na uloženie riadku do pamäte je použitý kontajner z STL, a to konktrétne `vector<string>`.

5.3.1 Výpočet hodnôt

Každý riadok reprezentuje 1 paket a sú z neho dopočítavané hodnoty, ktoré sú potom ukladané do RRD databázy pre neskoršie generovanie grafov. Zo zadania vyplýva, že aplikácia má vypočítavať a graficky zobrazovať jitter, kapacitu dátového toku, veľkosť paketov a časovú medzeru medzi paketmi.

Veľkosť paketu nie je nutné počítať, je medzi údajmi, ktoré táto časť aplikácie dostane na vstupe. Výpočet časovej medzery medzi paketmi je riešený jednoducho, a to tým, že si aplikácia pamätá časovú známku minulého paketu a potom ju odčíta od časovej známky súčasného.

Kapacita dátového toku je vlastne počet bajtov, ktorý bol prenesený v dátovom toku za sekundu. Rovnako teda prebieha aj výpočet tejto veličiny. Pri spracovaní každého paketu sa do premennej `timestamps` typu `list<struct timeval>`, kde `struct timeval` je štruktúra z knižnice jazyka

C `sys/time.h`, vloží na koniec časová známka paketu, ktorý sa práve spracuje. Do premennej `sizes` typu `list<int>` sa zase na koniec vloží veľkosť spracovávaného paketu. Táto veľkosť sa tiež pričíta k premennej `capacity` typu `int`, ktorá uchováva medzisúčet kapacity dátového toku. Následne sa pomocou funkcie `subtractTV()` zisťuje, či je rozdiel medzi prvou a poslednou hodnotou v premennej `timestamps` väčší ako 0 sekúnd, teda aspoň 1 sekunda. Ak áno, tak sa z premenných `timestamps` a `sizes` odoberú prvé hodnoty a od premennej `capacity` sa odčíta odoberaná veľkosť paketu. Tieto kroky sa opakujú, až kým podmienka spomínaná vyššie vyjde nepravdivá.

Ukážka 5.4: Výpočet kapacity dátového toku

```
while (subtractTV(timestamps.front(), timestamps.back()).
    tv_sec > 0) {
     timestamps.pop_front();
     capacity -= sizes.front();
     sizes.pop_front();
}
```

Jitter je v mojej aplikácii rozdiel medzi súčasnou časovou medzerou medzi paketmi a priemerom časových medzier v zadanom časovom úseku. Výpočet súčtu časových medzier prebieha takmer identicky ako výpočet kapacity dátového toku.

Ukážka 5.5: Výpočet jitteru

```
delay = subtractTV(last, current);
delays.push_back(delay.tv_sec + 0.000001 * delay.tv_usec);
sumDelays += delays.back();
while (subtractTV(jitterTimestamps.front(), jitterTimestamps
    .back()).tv_sec > atoi(argv[3]) - 1) {
     jitterTimestamps.pop_front();
     sumDelays -= delays.front();
     delays.pop_front();
}
jitter = delays.back() - sumDelays / delays.size();
```

5.3.2 Detekcia anomálií

Za anomáliu dátového toku v mojej aplikácii považujem to, keď bol prijatý rovnaký paket opakovane, alebo ak paket dorazil v nesprávnom poradí. Priebeh detekcie anomálií sa líši podľa toho, aký protokol bol použitý na transportnej vrstve paketu.

Pri detekcii sa používajú mapy, jedna pre protokol TCP a druhá pre UDP (de facto RTP). Keďže dátový tok sa dá jednoznačne určiť zdrojovou a cieľovou IP adresou paketu a zdrojovým a cieľovým portom paketu, kľúč na pristupovanie do tejto mapy je tvorený reťazcom zloženým z týchto hodnôt. Tieto mapy potom uchovávajú údaje o poslednom prijatom pakete z daného dátového toku.

Pri protokole TCP, ako jeho názov Transmission Control Protocol napovedá, je kontrola prenosu na strane protokolu, takže pri paketoch používajúcich protokol TCP táto časť aplikácie dostáva na vstupe aj sekvenčné a potvrdzovacie číslo a príznaky paketu, podľa ktorých detekcia anomálií prebieha. Pri protokole UDP je na vstupe sekvenčné číslo protokolu RTP, ktorý je jediný protokol s protokolom UDP na transportnej vrstve, pri ktorom moja aplikácia podporuje detekciu anomálií. Tieto údaje o prenesených paketoch sú ukladané do horeuvedených máp a pri vyhodnotení paketu ako anomálie sa inkrementuje počítadlo anomálií aj počítadlo paketov, inak sa inkrementuje len počítadlo paketov.

Informácia o počte anomálií a počte prenesených paketov sa v pravidelných intervaloch zobrazuje užívateľovi. Dĺžku tohto intervalu zadáva užívateľ. Toto zobrazovanie je riadené vláknom, ktoré je časované pomocou funkcie `sleep()`. Keďže je použité ďalšie vlákno musí byť prístup k počítadlám thread-safe, preto sú to globálne premenné a prístup k nim je možný len získaním zámku. Vlákno pri výpise zároveň vynuluje hodnoty oboch počítadiel.

5.3.3 Spolupráca s nástrojom RRDtool

Pri práci s nástrojom RRDtool a knižnicou librrd som využil hlavne funkcie pre vytvorenie RRD databázy (`rrd_create()`), vloženie dát do RRD databázy (`rrd_update_r()`) a generovanie grafov z RRD databázy (`rrd_graph()`). Kvôli lepšej prehľadnosti kódu som implementoval funkcie s výhodnejším rozhraním, ktoré tieto funcie obaľujú.

Funkcia `createRRD()`, viď 5.6, obaľuje funkciu `rrd_create()` a slúži na vytvorenie RRD databázy. Pri úspechu vracia hodnotu `true`, inak `false`. Ako argument táto funkcia potrebuje názov súboru, kde má byť RRD databáza umiestnená. Ak už databáza s daným menom existuje, nebude prepísaná kvôli použitiu prepínača -O v premennej `rrdparams`. Výsledkom bude RRD databáza, ktorá bude prijímať nové údaje každú sekundu. Taktiež bude mať 5 zdrojov dát, a to jitter, kapacitu dátového toku, počet prenesených paketov za sekundu a časovú medzeru medzi paketmi. Vytvorená RRD databáza bude tiež obsahovať 4 archívy, ktoré budú uchovávať hodnoty za

sekundu počas jednej minúty, minútu počas jednej hodiny, hodinu počas jedného dňa a dňa počas jedného týždňa.

Ukážka 5.6: Funkcia createRRD()

```
bool createRRD(const char* rrd) {
    char *rrdparams[] = {
        (char*) "rrdcreate",
        const_cast<char*>(rrd),
        (char*) "--step",
        (char*) "1",
        (char*) "-O",
        (char*) "DS:jitter:GAUGE:2:U:U",
        (char*) "DS:capacity:GAUGE:2:0:U",
        (char*) "DS:count:GAUGE:2:0:U",
        (char*) "DS:size:GAUGE:2:0:U",
        (char*) "DS:delay:GAUGE:2:0:U",
        (char*) "RRA:MAX:0.5:1:60",
        (char*) "RRA:AVERAGE:0.5:60:60",
        (char*) "RRA:AVERAGE:0.5:3600:24",
        (char*) "RRA:AVERAGE:0.5:86400:7",
        NULL
    };
    optind = opterr = 0;
    rrd_clear_error();
    int result = rrd_create(14, rrdparams);
    if (rrd_test_error() || (result != 0)) {
        cerr << "Error␣while␣creating␣RRD␣database:␣" <<
           rrd_get_error() << endl;
        return false;
    }
    return true;
}
```

Funkcia `updateRRD()`, viď 5.7, obaľuje funkciu `rrd_update_r()` a slúži na vkladanie dát do RRD databázy. Pri úspechu vracia hodnotu `true`, inak `false`. Táto funkcia je volaná po každom spracovanom pakete. Argumentami tejto funkcie sú časová známka paketu, hodnoty, ktoré sa budú vkladať do RRD databázy a názov súboru obsahujúceho RRD databázu.

Ukážka 5.7: Funkcia updateRRD()

```
bool updateRRD(const struct timeval& timestamp, const double
   & jitter, const int& capacity, const int& size, const int
   & count, const struct timeval& delay, const char* rrd) {
    string val = to_string(timestamp.tv_sec) + ":" +
       to_string(jitter) + ":" + to_string(capacity) + ":" +
        to_string(count) + ":" + to_string(size) + ":" +
```

```
        to_string(delay.tv_sec) + "." + to_string(delay.
        tv_usec);
    optind = opterr = 0;
    rrd_clear_error();
    const char* res = val.c_str();
    int result = rrd_update_r(rrd, NULL, 1, &res);
    if (rrd_test_error() || (result != 0)) {
        cerr << "Error while updating RRD database: " <<
            rrd_get_error() << endl;
        return false;
    }
    return true;
}
```

Na generovanie grafov z RRD databázy slúžia funkcie `graphXXXRRD()`, kde XXX znamená Jitter, Capacity, Size, Count alebo Delay. Táto funkcia obaľuje funkciu `rrd_graph()`.

V ukážke 5.8 je vidieť konktrétne funkciu `graphJitterRRD()`, ktorá slúži na generovanie grafov vývoja jitteru. Parametrom tejto funkcie je názov RRD databázy, z ktorej sa má graf generovať. Grafy, ktoré táto funkcia generuje, zobrazujú vývoj danej veličiny za posledných 60 sekúnd (parameter `-s -60` v premennej `rrdparams`).

Generovanie grafov je spúšťané automaticky každú sekundu pomocou časovaného vlákna obdobným spôsobom ako pri výpise počtu anomálií. Aby nedochádzalo ku konfliktom medzi generovaním grafov a vkladaním údajov do RRD databázy, bol použitý zámok, ktorý nedovolí, aby funkcie `updateRRD()` a `graphXXXRRD()` bežali naraz.

Ukážka 5.8: Funkcia graphJitterRRD()

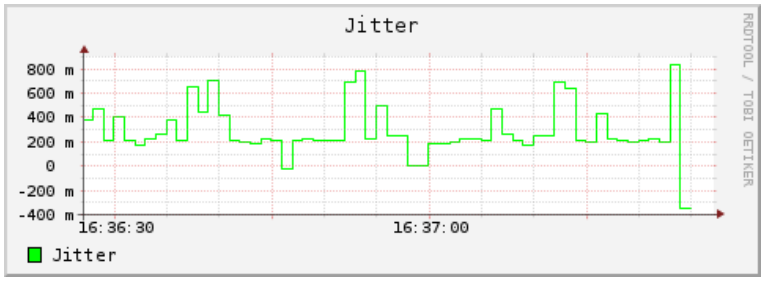
```
bool graphJitterRRD(const char* rrd) {
    char **ptr = NULL;
    int xsize, ysize, result;
    double xmax, ymax;
    stringstream ss;
    ss << "DEF:jitter=" << rrd << ":jitter:AVERAGE";
    string par = ss.str();
    char *rrdparams[] = {
        (char*) "rrdgraph",
        (char*) "jitter.png",
        (char*) "-s",
        (char*) "-60",
        (char*) "-a",
        (char*) "PNG",
        (char*) "--title",
        (char*) "Jitter",
        const_cast<char*>(par.c_str()),
```

```
        (char*) "LINE1:jitter#00FF00:Jitter",
        NULL
    };
    optind = opterr = 0;
    rrd_clear_error();
    result = rrd_graph(10, rrdparams, &ptr, &xsize, &ysize,
       NULL, &xmax, &ymax);
    ... //osetrenie chyby
}
```

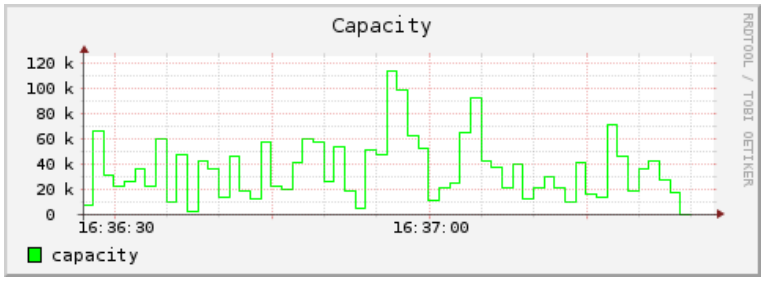
5.4 Grafická prezentácia

Aplikácia výsledky svojich meraní a následných výpočtov zobrazuje v grafoch pomocou jednoduchej webstránky. Na nej beží jednoduchý skript napísaný v JavaScripte, ktorý každú sekundu k ceste ku grafom pridá query string s novou náhodnou hodnotou dummy premennej, čím prehliadač donúti ich obnoviť, teda načítať novovygenerované grafy.

Obr. 5.1: Ukážka vygenerovaného grafu jitteru



Obr. 5.2: Ukážka vygenerovaného grafu kapacity dátového toku



Kapitola 6

Inštalácia, konfigurácia a použitie

V tejto kapitole, ako jej názov napovedá, sa budem venovať spôsobu inštalácie implementovanej aplikácie, možnostiam jej konfigurácie a radám na jej použitie.

6.1 Inštalácia

Pred samotnou inštaláciou je nutné overiť si, či systém, kde bude aplikácia inštalovaná, má nainštalované nasledujúce nástroje:

- kompilátor jazyka C++ podporujúci aspoň štandard C++11,
- Python aspoň verzie 3,
- nástroj RRDtool a knižnicu librrd na prácu s ním,
- knižnicu libpcap,
- na prípadné zobrazenie generovaných grafov webový prehliadač podporujúci JavaScript.

Aplikácia bude dodaná v komprimovanom archíve, ktorý bude obsahovať inštalačný skript. Ten skompiluje prítomné zdrojové súbory aplikácie do priečinka `/opt/video-signal-monitor`, skopíruje do neho spúšťací skript a vytvorí naň symbolický link v priečinku `/usr/bin` s názvom `video-signal-monitor`. Taktiež sa tam bude nachádzať webová stránka, ktorá bude graficky prezentovať výsledky činnosti aplikácie. V prípade neúspechu pri inštalácii vypíše skript inštrukcie, ako nainštalovať aplikáciu ručne.

6.2 Konfigurácia

Spúšťací skript implementovanej aplikácie ponúka možnosti konfigurácie prostredníctvom týchto prepínačov a parametrov:

- **-s/--sniff** *meno_súboru*: Spustí sa iba odchytávanie paketov. Informácie potrebné pre ďalšie spracovanie budú uložené do zadaného súboru.
- **-p/--process** *meno_súboru*: Spustí sa iba spracovanie paketov zo zadaného súboru vytvoreného spustením aplikácie s parametrom -s.

Tieto 2 parametre sú vzájomne nezlúčiteľné a ak nie je použitý ani jeden z nich, pri spustení aplikácie prebieha súčasne odchytávanie paketov aj ich spracovanie.

- **-i/--interface** *interface*: Názov sieťového rozhrania, z ktorého sa budú odchytávať pakety. Ak nebude rozhranie zadané, bude použité rozhranie získané aplikáciou.
- **--rtp**: Aplikácia bude odchytávať iba pakety, ktoré vyhovejú detekcii RTP hlavičky (viď 5.2).
- **--tcp**: Aplikácia bude odchytávať iba pakety používajúce protokol TCP na transportnej vrstve.
- **--src** *IP_adresa*: Zdrojová IP adresa paketov.
- **--sport** *číslo_portu*: Číslo zdrojového portu.
- **--dport** *číslo_portu*: Číslo cieľového portu.

Horeuvedené prepínače a parametre nie je možné použiť, ak aplikácia iba spracuje už odchytené pakety. Prepínače --rtp a --tcp sú vzájomne nezlúčiteľné.

- **-r/--rrd** *meno_súboru*: Umiestnenie RRD databázy. Ak súbor neexistuje, bude vytvorená nová RRD databáza.
- **-o/--outfile** *meno_súboru*: Meno súboru, do ktorého sa bude ukladať log paketov podľa zadania bakalárskej práce.
- **-a/--anomalyTime** *interval*: Časový úsek (v sekundách), v ktorom sa budú vyhodnocovať anomálie v dátovom toku. Východzia hodnota je 5 sekúnd.

- **-j/--jitterTime** *interval*: Časový úsek (v sekundách), v ktorom sa bude vyhodnocovať jitter. Východzia hodnota je 1 sekunda.

Horeuvedené parametre nie je možné použiť, ak bola aplikácia spustená len s odchytávaním paketov. Parametre -r a -o sú povinné, ak aplikácia spracuje pakety alebo prebieha odchytávanie a spracovanie naraz.

6.3 Použitie

Nainštalovaná aplikácia sa spúšťa príkazom `video-signal-monitor`, ktorý je konfigurovateľný prepínačmi a parametrami z predošlej podkapitoly. Aplikáciu je potrebné spustiť s právami roota. Je to vyžadované kvôli odchytávaniu paketov. V reálnom čase je potom možné sledovať vývoj sledovaných veličín popisujúcich dátový tok na webovej stránke umiestnenej v `/opt/video-signal-monitor/monitor.html`.

Pre čo najpresnejšie výsledky meraní je potrebné vedieť identifikovať dátový tok, ktorý chceme analyzovať. Na identifikáciu dátového toku nám môže poslúžiť znalosť protokolu, ktorý je použitý na prenos videa, prípadne znalosť portov, ktoré daný protokol využíva. Ak nám tieto údaje nie sú známe, môžeme si pomôcť nejakým nástrojom na všeobecné odchytávanie paketov ako `tcpdump` alebo `wireshark`.

Keďže odchytávanie paketov prebieha vo funkcii `pcap_loop()` v nekonečnom cykle, je nutné aplikáciu ukončiť zaslaním signálu SIGTERM (klávesová skratka v termináli Ctrl + C). V aplikácii je implementovaný handler, ktorý sa postará o jej následné regulérne ukončenie.

Kapitola 7

Testovanie výslednej aplikácie

V poslednej kapitole sa venujem procesu otestovania výslednej aplikácie, pri ktorom bola overená jej funkčnosť na rôznych operačných systémoch, jej správanie sa pri kontrole prenosu videa rozličnými technológiami a zobrazovanie ňou vygenerovaných grafov.

7.1 Kompatibilita s operačnými systémami

Počas vývoja a testovania prišla aplikácia do styku s týmito operačnými systémami:

- Ubuntu 15.10 Wily Werewolf
- Xubuntu 16.04 Xenial Xerus
- Fedora 23

7.2 Testy podľa použitej technológie prenosu

Test každej technológie prebiehal približne 30 minút, aby sa overilo správanie pri rastúcich výstupných súboroch. Popri odchytávaní paketov mojou aplikáciou boli pakety odchytávané aj aplikáciou Wireshark, ktorá v tomto prípade slúži na kontrolu správnosti analýzy paketov. Počas testov boli do aplikácie pridané debugovacie výpisy, ktoré slúžili na porovnanie s výsledkami z Wiresharku.

Pri testoch bol použitý počítač s nasledujúcimi parametrami:

- **Procesor**: Intel Celeron J1800, 2 jadrá s frekvenciou 2,4GHz.
- **Pamäť RAM**: 4GB.
- **HDD**: 1000 GB.
- **Rýchlosť sieťovej karty**: 1 Gbps.

7.2.1 RTP

Protokol RTP bol testovaný na streamovanej televízii dostupnej na sieti na Koleji Podolí. Na testovanie bol použitý kanál ČT4 HD s rozlíšením 1920x1080 pixelov. Táto streamovaná televízia je dostupná v programe VLC media player [22], ktorý na jej príjem používa port 1234, preto pri odchytávaní bola aplikácia spustená s príslušným filtrom.

Počas testu aplikácia odchytila 1 028 596 paketov, z čoho 35 vyhodnotila ako anomálie. Pri skúmaní testovacieho výpisu som ale zistil chybu pri vyhodnocovaní anomálie spočívajúcu v tom, že pri implementácii detekcie anomálií som nepočítal s vynulovaním sekvenčného čísla hlavičky protokolu RTP po pretečení. Táto chyba je už odstránená a po odčítaní chybne označených anomálií ich v konečnom dôsledku nastalo iba 20.

7.2.2 RTMP

Pre testovanie protokolu RTMP bolo použité živé vysielanie slovenskej televízie Joj [23] v rozlíšení 1280x720 pixelov. Protokol RTMP používa port 1935, rovnaký port preto ako zdrojový port paketov používala aj moja aplikácia.

Pri testovaní aplikácia odchytila 144 081 paketov, z čoho bolo 705 označených ako anomálie. Pri porovnaní so zisteniami Wiresharku som zistil, že počty anomálií si približne odpovedajú, pričom moja aplikácia va každom zhluku anomálií občas označí ako anomáliu o jeden paket navyše ako Wireshark.

Počas prezerania debugovacích výpisov som taktiež narazil na chybu pri logovaní časovej známky, ktorú som opravil.

7.2.3 HDS

Technológiu HDS v archíve svojho vysielania používa slovenská RTVS [24]. Z tohto dôvodu som tento archív využil pri testovaní mojej aplikácie. Technológia HDS (HTTP Dynamic Streaming), ako jej názov napovedá, používa

na prenos videa protokol HTTP, preto pri tomto teste som aplikáciu spustil s filtrom, ktorý prijímal len pakety so zdrojovým portom 80. Na testovanie som použil video s rozlíšením 1280x720 pixelov.

Pri približne 30 minút trvajúcom teste bolo aplikáciou odchytených 88 615 paketov. Počet anomálií odhalených implementovanou aplikáciou je 215, Wireshark ich odhalil 246. Podobne ako pri teste RTMP, aj pri tomto teste moja aplikácia miestami v zhlukoch anomálií niekde označila menej, inde viac paketov ako anomáliu, vo výsledku to však nemalo príliš veľký dopad.

7.2.4 HLS

Stránka iVysílání [25] od Českej televízie pre prenos videa používa technológiu HLS (HTTP Live Streaming), ktorá podobne ako HDS, používa na prenos videa protokol HTTP. Z tohto dôvodu som ju použil pri teste a aplikáciu spustil s filtrom prijímajúcim pakety so zdrojovým portom 80.

Počas testu aplikácia odchytila 49 545 paketov a detegovala v nich 482 anomálií. Toto číslo je pomerne vysoké a znamená, že necelé 1 % odchytených paketov sú anomálie. Výsledky paketovej analýzy Wiresharku však k podobnému záveru dospeli tiež, takže sa nejedná o chybu na strane implementovanej aplikácie.

7.2.5 MPEG-DASH

Pre testovanie technológie MPEG-DASH bola použitá služba YouTube [26], ktorá ju využíva. Kedže YouTube používa protokol HTTPS, aplikácia bola spustená s filtrom, ktorý prepúšta pakety so zdrojovým portom 443. Na testovanie bolo použité video s rozlíšením 1920x1080 pixelov s frameratom 60 fps.

Počas približne 30 minútového testu bolo zachytených 116 518 paketov a aplikácia označila 222 paketov ako anomálie v dátovom toku, podobne ako kontrolná analýza paketov z Wiresharku.

7.2.6 Zhrnutie výsledkov

Aplikácia vyšla pozitívne z testov zo všetkými technológiami prenosu videa, ktoré boli testované. Počas testov ani raz nespadla a výsledky detekcie anomálií sú porovnateľné s výsledkami paketovej analýzy aplikácie Wireshark. Výsledky týchto testov sú uložené aj na priloženom CD.

7.3 Test zobrazenia vygenerovaných grafov

Webová stránka zobrazujúca grafy funguje podľa očakávaní. Tým, že obnovuje grafy každých 1,5 sekundy, sa vytvára imitácia animácie týchto grafov. Narazil som však na 2 problémy pri tomto zobrazovaní.

Prvým problémom je občasná absencia niektorého z grafov na stránke. Pravdepodobne nastáva ak webová stránka chce obnoviť graf a v tom okamihu je ten daný graf generovaný aplikáciou. Tento problém sa mi ani napriek dlhému snaženiu nepodarilo odstrániť.

Ďalším problémom je nemožnosť zvolenia kroku kratšieho ako 1 sekunda v nástroji RRDtool a tým spôsobená nemožnosť vkladať do RRD databázy údaje o každom odchytenom pakete. Tým pádom môžu byť grafy aj obsah RRD databázy skreslené oproti realite. Toto skreslenie bude najvýraznejšie viditeľné pri veličinách naviazaných na jednotlivé pakety, teda pri časovej medzere medzi paketmi, jitteri a veľkosti paketu.

Záver

V tejto práci som sa snažil priblížiť spôsoby prenosu videa cez internet, nástroje v súčasnoti používané na kontrolu tohto prenosu a technológie, ktoré ju umožňujú.

Ďalej som analyzoval, implementoval a otestoval aplikáciu umožňujúcu takúto kontrolu vykonávať. Aplikácia počas svojho behu užívateľa upozorňuje na anomálie v dátovom toku a pomocou webovej stránky mu zobrazuje grafy veličín popisujúcich kvalitu dátového toku. Tieto grafy sú generované v nástroji RRDtool, ktorý slúži zároveň ako round robin databáza, v ktorej sú údaje uložené.

V budúcnosti by bolo možné aplikáciu rozšíriť o grafické užívateľské rozhranie, ktoré by mohlo zjednodušiť prácu s aplikáciou a taktiež odstrániť problémy s občasným miznutím grafov zo súčasnej webovej stránky. Ďalej by bolo možné nájsť náhradu za RRDtool, ktorá nemá časové obmedzenie na vkladanie do databázy. Nakoniec by bolo možné popracovať na detekcii anomálií, ktorá síce v súčasnosti dáva relatívne presné výsledky, stále nie je dokonalá.

Literatúra

[1] Riiser, H.: *Adaptive Bitrate Video Streaming over HTTP in Mobile Wireless Networks*. Dizertační práce, Univerzita v Osle, Fakulta matematiky a prírodných vied, 2013.

[2] Daoust, F.; aj.: *Towards Video on the Web with HTML5*. 2010, [online], [cit. 09-05-2016]. Dostupné z: `https://www.w3.org/2010/Talks/1014-html5-video-fd/video-html5.pdf`

[3] Rao, A.; Legout, A.; Lim, Y.-s.; aj.: Network Characteristics of Video Streaming Traffic. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA: ACM, 2011, ISBN 978-1-4503-1041-3, s. 25:1–25:12, doi:10.1145/2079296.2079321. Dostupné z: `http://doi.acm.org/10.1145/2079296.2079321`

[4] Cisco Systems, Inc.: Cisco Visual Networking Index: Forecast and Methodology, 2014-2019 White Paper. [online], [cit. 09-05-2016]. Dostupné z: `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html`

[5] Schulzrinne, H.; Casner, S.; Frederick, R.; aj.: *RFC 3550 - RTP: A Transport Protocol for Real-Time Applications*. Internet Engineering Task Force, 2003.

[6] Popelka, V.: *Technologie pro streaming videa*. Diplomová práce, České vysoké učení technické v Prahe, Fakulta elektrotechnická, 2013.

[7] Medzinárodná organizácia pre normalizáciu: *ISO/IEC 23009-1 Information technology — Dynamic adaptive streaming over HTTP (DASH)*. 2014.

[8] McCanne, S.; Jacobson, V.: The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, USENIX'93, Berkeley, CA, USA: USENIX Association, 1993, s. 2–2. Dostupné z: http://www.tcpdump.org/papers/bpf-usenix93.pdf

[9] Protocog: Q&A with the founder of Wireshark and Ethereal. [online], [cit. 09-05-2016]. Dostupné z: http://www.protocog.com/gerald_combs_interview.html

[10] Dotcom-Monitor: Media Server and Streaming Video Monitoring. [online], [cit. 09-05-2016]. Dostupné z: https://www.dotcom-monitor.com/server-monitor/media-server-streaming-video-monitoring/

[11] Dotcom-Monitor: Media Streaming Task. [online], [cit. 09-05-2016]. Dostupné z: https://wiki.dotcom-monitor.com/monitoring-platforms/serverview/media-streaming-task/

[12] SciVisum Ltd.: Media Streaming Monitoring. [online], [cit. 09-05-2016]. Dostupné z: http://www.scivisum.co.uk/services/media-streaming-monitoring/

[13] Radcom Ltd.: Media Streaming Monitoring Solution. [online], [cit. 09-05-2016]. Dostupné z: http://radcom.com/media-streaming-monitoring-solution

[14] Stroustrup, B.: FAQ. [online], [cit. 12-05-2016]. Dostupné z: http://www.stroustrup.com/bs_faq.html

[15] Hundt, R.: Loop Recognition in C++/Java/Go/Scala. In *Proceedings of Scala Days 2011*, 2011, [online], [cit. 12-05-2016]. Dostupné z: https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf

[16] Garcia, L. M.: Programming with Libpcap - Sniffing the Network From Our Own Application. *Hakin9*, 2008, [online], [cit. 12-05-2016]. Dostupné z: http://recursos.aldabaknocking.com/libpcapHakin9LuisMartinGarcia.pdf

[17] van den Bogaerdt, A.: RRDtool - rrdtutorial. [online], [cit. 12-05-2016]. Dostupné z: `http://oss.oetiker.ch/rrdtool/tut/rrdtutorial.en.html`

[18] Oetiker, T.; Stamfest P.: RRDtool - rrdcreate. [online], [cit. 12-05-2016]. Dostupné z: `http://oss.oetiker.ch/rrdtool/doc/rrdcreate.en.html`

[19] Severance, C.: JavaScript: Designing a Language in 10 Days. *Computer*, ročník 45, č. 2, 2012: s. 7–8, ISSN 0018-9162, doi:http://doi.ieeecomputersociety.org/10.1109/MC.2012.57.

[20] Venners, B.: The Making of Python - A Conversation with Guido van Rossum, Part I. [online], [cit. 12-05-2016]. Dostupné z: `http://www.artima.com/intv/pythonP.html`

[21] Cotton, M. S.; Eggert, L.; Touch, J. D.; aj.: *RFC 6335 - Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry*. Internet Engineering Task Force, 2011.

[22] VideoLAN: Official page for VLC media player, the Open Source video framework! [online], [cit. 15-05-2016]. Dostupné z: `http://www.videolan.org/vlc/`

[23] MAC TV s.r.o.: Live vysielanie. [online], [cit. 15-05-2016]. Dostupné z: `http://live.joj.sk/`

[24] Rozhlas a Televízia Slovenska: Televízny videoarchív. [online], [cit. 15-05-2016]. Dostupné z: `http://www.rtvs.sk/televizia/archiv`

[25] Česká televize: iVysílání. [online], [cit. 15-05-2016]. Dostupné z: `http://www.ceskatelevize.cz/ivysilani/`

[26] YouTube LLC: YouTube. [online], [cit. 15-05-2016]. Dostupné z: `https://www.youtube.com/`

Dodatok **A**

Zoznam použitých skratiek

AES Advanced Encryption Standard

BPF BSD Packet Filter

BSD Berkeley Software Distribution

CSPF CMU/Stanford Packet Filter

CSV Comma-separated values

DASH Dynamic Adaptive Streaming over HTTP

DRM Digital rights management

HDS HTTP Dynamic Streaming

HLS HTTP Live Streaming

HTTP Hypertext Transfer Protocol

ISO International Organization for Standardization (Medzinárodná organizácia pre normalizáciu)

KPI Key Performance Indicator (kľúčový výkonnostný ukazateľ)

MMS Microsoft Media Server

MPD Media Presentation Description

MPEG Moving Picture Experts Group

NIT Network Interface Tap

QoE Quality of Experience (kvalita vnímania)

QoS Quality of Service (kvalita služby)

RFC Request for Comments

RRD Round robin database

RTMP Real Time Messaging Protocol

RTP Real-Time Transport Protocol

RTSP Real Time Streaming Protocol

SIP Session Initiation Protocol

SRTP Secure Real-Time Transport Protocol

STL Standard Template Library

TCP Transmission Control Protocol

UDP User Datagram Protocol

VoIP Voice over IP

Dodatok **B**

Obsah priloženého CD

- `src` ..... zdrojové súbory implementácie
- `thesis`
  - `tex` ..... zdrojová forma práce vo formáte LaTeX s ďalšími súbormi
  - `BP_Beneš_Jozef_2016.pdf` ..... text práce vo formáte PDF
- `readme.txt` ..... informácie o tomto CD
- `results.rar` ..... archív s výsledkami testov
- `video-signal-monitor.tar.gz` ..... inštalačný balíček implementácie