Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

IP chůvička

Plzeň 2014 Petra Křivanová

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 9. května 2014

Petra Křivanová

Abstract

IP Baby Monitor

This thesis presents a development of a baby monitor (hereinafter "monitor") which can use a computer network. Many of the commonly used monitors have a considerable disadvantage - they do not work in a room where the WiFi is established. This problem is caused by the fact that respective devices use radio waves with the same frequency. In addition, monitors usually do not support any kind of protection for data transmission. This shortage does not only allow the intruder to watch the baby and the equipment yet he can even spy the people.

The main goal is to create a baby monitor which can use the computer network (WiFi) for encrypted data transmission of audio and video streams. This IP baby monitor is a software application that is implemented for two operating systems (Windows, GNU/Linux). The monitor uses the server-client architecture. The server is a program running on a device with camera and microphone and it is placed in the baby's room. The client is in the parent's room and it can play the stream produced by the server.

The server program can also analyze the stream from the camera and the sound from the microphone and it can find motion or volume changes. The clients are notified with a message if there is such a change. The parents have continual supervision of what happens in baby's room.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Dětské elektronické chůvičky (dále jen chůvičky) již dávno nejsou jen jednocestné vysílačky pro přenos zvuku. Dnes jsou k dostání chůvičky, které kromě přenosu zvuku zvládají i přenos obrazu, hlídání teploty v místnosti, hlídání hladiny zvuku, detekují změnu obrazu, hrají dětem ukolébavky nebo mají i noční vidění. Hodně dnešních chůviček má však jednu zásadní vadu, a sice že vysílají ve stejném frekvenčním pásmu, jako WiFi sítě. Výsledek je takový, že ani jedno zařízení nefunguje tak, jak by mělo. V některých případech dokáží chůvičky dokonce i vyřadit bezdrátovou síť z provozu úplně.

Cílem této práce je vytvořit softwarovou aplikaci, která bude používat standardní síťové protokoly a bude mít takové funkce, aby ji bylo možné použít jako náhradu stávajících chůviček. Aplikace by tedy měla umět přenášet zvuk, video, detekovat změnu obrazu nebo zvuku a dát o těchto změnách, nějakým způsobem, vědět příjemci. Oproti běžným chůvičkám bude aplikace provádět šifrování veškerého datového přenosu. Díky použití standardních síťových protokolů je možné využít existujících drátových i bezdrátových síti k přenosu jak videa tak zvuku. Tím se eliminuje nežádoucí rušení, které se u běžných chůviček jinak objevuje.

2 Existující řešení

2.1 Teoretický základ

2.1.1 Stávající řešení

Dětská chůvička je malé elektronické zařízení, které se používá pro hlídání a monitorování malých dětí. Obvykle se skládá ze dvou částí - vysílací a přijímací. Vysílací část je umístěna v pokoji dítěte, zachytává zvuk a odesílá ho na přijímací zařízení, které je v pokoji rodičů a oni díky tomu slyší, že se něco děje. Na takovémto principu fungují všechny chůvičky. Dnes jsou k dostání i chůvičky, které umožňují dvoucestnou komunikaci, tedy je možné na dítě přes přijímací zařízení mluvit. Mimo to existují i chůvičky s přenosem obrazu nebo detekcí změny hladiny zvuku. Součástí těch lepších je i infra snímání okolí a další funkce. Objevily se už i chůvičky, které dokáží pro přenos dat používat WiFi síť.

2.1.2 Nová aplikace

Nově vytvořená aplikace vychází z konceptu fungování stávajících chůviček, čili je taktéž rozdělena na část přijímací (dále jen klient) a část vysílací (dále jen server). Jedná se o softwarovou aplikaci, která byla vytvořena především pro použití v privátních sítích[1], a kde je nutné, aby na zařízení, na kterém běží server, byla umístěna kamera a mikrofon, na klientovi zobrazovací zařízení a jak klient, tak server měli přístup do počítačové sítě. Klient si může nechat od serveru posílat aktuálně snímaný zvuk a video (dále stream) ze vstupních zařízení a nechat si posílat zprávy při zvýšení hladiny zvuku nebo pohybu před kamerou. Dále je možné příchozí stream u klienta ukládat na disk. Aplikace neumožňuje obousměrné posílání streamu, tzn. na dítě není možné přes aplikaci mluvit. Server podporuje připojení více klientů najednou a veškerá komunikace mezi serverem a klienty je šifrovaná, včetně přenosu streamu.

[1] Aplikace funguje i přes internet, pokud má zařízení se serverem přidělenu veřejnou IPv4 adresu.

2.2 Výhody a nevýhody

2.2.1 Stávající řešení

Jak již bylo řečeno, hlavní důvod, proč tato aplikace vznikla, je ten, že digitální chůvičky využívají ke svým přenosům stejné frekvenční pásmo jako WiFi sítě. Kvůli tomu jsou obě zařízení navzájem rušena nebo nefungují vůbec. To představuje obrovský problém hlavně pro byty v panelákových domech, kde je na malém prostoru seskupeno velké množství WiFi sítí. Většina chůviček má navíc zabudovaný jen jeden kanál, na kterém vysílá a není možné ho změnit. Mezi další nebezpečné záležitosti velkého počtu chůviček patří to, že nevarují, pokud dojde ke ztrátě signálu.

U analogových chůviček nedochází sice k tomu, že by se rušily s WiFi sítí, ale je tu jiný závažný problém a to narušení soukromí, a tím i bezpečnosti. U těchto typů chůviček se často stává, že dítě, které je slyšet, nepatří těm rodičům, kteří ho slyší. Kdokoliv na stejné frekvenci může poslouchat.

Na trhu lze dostat i několik málo chůviček, které využívají WiFi síť. Opět však nepodporují šifrování přenosu a některé jsou navíc špatně provedené. Video nebo zvuk není tak kvalitní, jak je uvedeno ve specifikacích, nebo chůvičky neposkytují připojení více odběratelů videa/zvuku. Zařízení buď více odběratelů najednou nepodporují vůbec nebo podporují špatně. Všichni uživatelé mají stejné nastavení a vzájemně si ho pod rukou mohou měnit[2].

Podle úhlu pohledu by bylo možné za výhodu prohlásit to, že chůvičky nepotřebují ke svému fungování WiFi. U většiny chůviček se také uvádí větší dosah (v metrech), než který nabízí WiFi. Na chůvičku pracující přes Internet je ale možné se připojit odkudkoliv, na rozdíl od běžné chůvičky, jejíž dosah končí cca kolem $200m$. Běžná chůvička bude fungovat i při vypnutí elektrického proudu, pokud je na baterky a ty jsou nabité. Souhrn nejčastějších nedostatků současných chůviček je v tabulce 2.1.

[2]V době, kdy již byla tato práce zadána, vzniklo několik nových chůviček, které, kromě jiného, podporují jak šifrování přenosu, tak používají WiFi síť, a obraz se dá zobrazit na chytrých telefonech.

#	Nedostatky stávajících chůviček
1	Není možný paralelní provoz chůvičky s WiFi.
2	Nešifrovaný přenos.
3	Rušení jinými chůvičkami.
4	Neupozornění při ztrátě signálu.
5	Nemožnost výměny pouze kamery/mikrofonu.
6	Není možnost připojit více přijímačů.
7	Nelze měnit nastavení posílaného streamu.
8	Omezený dosah.
9	Není možnost výběru, zda sledovat video nebo audio.
10	Nelze ukládat obraz/zvuk do souboru.

Tab. 2.1: Souhrn nedostatků současných chůviček.

2.2.2 Nové řešení

Tím, že aplikace využívá již existující rozhraní pro komunikaci, kterým jsou počítačové sítě, řeší největší problém chůviček. Dalším velký problémem, který je zde vyřešen, je bezpečnost. Jednak v podobě ověřování přihlašovacích údajů na straně serveru, a pak také šifrováním veškerého přenosu včetně šifrování streamů. Další výhodu představuje i to, že k serveru se najednou může připojit více klientů, kteří mohou sledovat stream a mít svoje vlastní nastavení, co se týče zasílání upozornění nebo i typu zasílaného streamu (audio/video). Obraz je možné sledovat buď nepřetržitě nebo si ho nechat pustit vždy v případě nějakého podnětu (šum, pohyb).

Na rozdíl od ostatních chůviček je možné si vybrat, zda sledovat video se zvukem nebo jen zvuk[3]. Tato možnost je výhodná v případě pomalého připojení, které by nemuselo velký objem dat, který představuje přenos video streamu, propustit. Mezi další věci, které chůvičky neobsahují, patří nastavení kvality přenášeného streamu. Upravit lze jak zvuk, tak video. Samozřejmě, toto nastavení pak sdílí všichni klienti.

V případě, že klient ztratí spojení se serverem, je o tom uživatel ihned informován.

Mezi nevýhody lze řadit to, že aplikaci je nutné spustit na zařízení, které podporuje Windows/Linux a má funkční kameru a mikrofon. To platí pro server, klient

[3]Pokud nebude uvedeno jinak, budou přenášené streamy nadále označovány jako video (video+audio) a audio.

musí být naopak puštěn na zařízení se zobrazovacím zařízením a reproduktory. V případě, že rodič pracuje z domova, může mít na počítači puštěného klienta, a v případě, že se něco začne dít, je upozorněn zprávou. Díky tomu nemusí sledovat více zařízení najednou. Upozornění je pouze vizuální, aplikace nepodporuje žádné audio alarmy ani vibrace. V současné době aplikace, ve výchozím nastavení, počítá s použitím jedné kamery a jednoho mikrofonu. Při použití uživatelského nastavení by bylo možné snímat obraz ze dvou kamer. Omezení je dáno tím, že pouze na dvou portech poslouchají udp servery, které streamy šifrují a poskytují klientům.

Aplikace neumožňuje obousměrnou komunikaci. To znamená, že na dítě není možné pomocí aplikace mluvit. Zda se jedná o výhodu, či nevýhodu, je sporné. Pokud dítě začne brečet, stejně se na něj rodič bude muset jít podívat. Navíc, dítě by mohlo mást, že na něj někdo mluví, ale nikoho nevidí.

2.2.3 Souhrn

Nyní se podíváme na shrnutí toho, jaké problémy mívají současné chůvičky, a které z nich a jak řeší nová aplikace. Hlavní problém, jak už bylo několikrát zmíněno, je nefunkčnost chůviček společně s WiFi. Chůvičky, které vysílají ve stejném frekvenčním pásmu jako WiFi sítě, jsou rušeny a nefungují správně nebo vůbec. Hodně chůviček má navíc předvolen jen jeden kanál pro přenos dat. Aplikace tento závažný problém řeší tak, že k přenosu dat využívá počítačové sítě a standardní síťové protokoly. Je pak nutné, aby obě zařízení s aplikací měla připojení do počítačové sítě.

Hodně chůviček stále nepodporuje šifrování přenášených dat. Data je možné bez problémů odposlouchávat a narušovat tak soukromí osob. Hlavní nebezpečí pak představují video chůvičky, kde útočník může sledovat např. vybavení bytu. Existují i chůvičky, které též přenáší data přes počítačovou síť a jediné zabezpečení představuje ověření uživatele pomocí jména a hesla. Data šifrována nejsou. Aplikace proto šifruje veškerý přenos dat. Šifrovány jsou i přenášené streamy.

Chůvičky jsou většinou tvořeny z jednoho vysílače a jednoho přijímače. Sledovat dítě, tak může vždy jen jedna osoba. Aplikace podporuje připojení více klientů, kteří mohou odebírat stream. Dítě tak může kontrolovat každý, kdo má přístup do aplikace.

Chůvičky většinou slouží k tomu, aby přepravily zaznamenaný vstup na přijímací zařízení a už neřeší nastavení vlastností přenášených dat. Není tak například možné upravit velikost nebo kvalitu přenášeného videa. U existujících video chůviček, které používají počítačové sítě, to může být problém např. při přetížení sítě. Přednastavená kvalita videa představuje příliš velký objem dat, který není možné přenést. Nová chůvička umožňuje nastavení parametrů přenášených streamů. Je tak možné nastavit počet audio kanálů, velikost nebo kvalitu videa a audio frekvenci. Díky tomu lze hodně redukovat objem přenášených dat.

Další problém, který s tímto souvisí, je nemožnost vybrání si sledovaného streamu. Pokud video chůvička přenáší i zvuk, většinou zde není na výběr, jestli bude uživatel sledovat obraz se zvukem nebo jen zvuk. V případě vytížení sítě by se opět hodila možnost mít na výběr a sledovat jen audio, které je oproti videu podstatně méně náročné na objem přenesených dat. Aplikace tuto vlastnost má a poskytuje uživatelům 2 možnosti. Buď mohou sledovat video se zvukem, nebo samotný zvuk.

V nedávné době se objevila nová chůvička, která podporuje snímání obrazu z více kamer najednou. Jedná se o užitečnou vlastnost v případě, že jsou v pokoji umístěny např. dvě děti nebo dítě je už větší, a mohlo by se dostat ze zorného úhlu kamery, která nedokáže snímat celý pokoj. Jedná se o novou funkcionalitu a je pouze pár chůviček, které jí poskytují. Naše aplikace počítá s použitím jedné kamery a jednoho mikrofonu. Má však i funkci pro zadání vlastního příkazu, který získává vstupní streamy. Tím je umožněno zkušeným uživatelům použití i dvou kamer najednou.

Hodně chůviček obsahuje upozornění v případě nějaké události. Upozornění může být buď zvukové, vizuální nebo vibrační. Mezi nejčastější události pak patří pohyb před kamerou nebo zvýšení hlasitosti v pokoji dítěte. Naše aplikace využívá pouze vizuálních upozornění v podobě zpráv v oznamovací oblasti.

Častým a i nebezpečným problémem je, že pokud se přijímač dostane mimo dosah vysílače, nedá o tom žádným způsobem vědět uživateli. Ten si pak myslí, že se u dítěte nic neděje, ale skutečnost může být jiná. Některé chůvičky to řeší např. varovným signálem. Nová aplikace tento problém též řeší. Pokud dojde ke ztrátě spojení mezi serverem a klientem, klient je upozorněn zprávou a jeho program je ukončen. Ke ztrátě spojení může dojít buď důsledkem chyby při přenosu dat nebo i tak, že server byl odpojen.

Hodně chůviček poskytuje kameru s nočním viděním. Dítě je tak možné bez problémů sledovat i v noci. Ne všechna zařízení jsou ale natolik kvalitní, aby na nich bylo v noci něco vidět. Naše aplikace přenáší stream bez ohledu na to, o jakou kameru se jedná. Pokud bude připojena infra kamera, bude se přenášet stream z infra kamery. Tím pádem záleží pouze na uživateli, jakou kameru si pořídí. V případě, že kamera nebude kvalitní nebo se rozbije, lze ji, na rozdíl od běžných chůviček, bez problémů vyměnit.

Chůvičky se vytváří tak, aby byly mobilní a tudíž jsou převážně napájené na baterie. U naší aplikace záleží na tom, na jakých zařízeních jednotlivé programy poběží. Server je možné spustit na malém zařízení s procesorem ARM, které je též možné napájet z baterie.

Jako přednost některé chůvičky uvádí možnost mluvení na dítě přes zařízení. Naše aplikace toto nepodporuje. Dítě by mohlo mást, pokud by na něj někdo mluvil a ono by ho nevidělo. Pravděpodobně by pak utěšování skrz chůvičku ani nefungovalo. Lepší vlastností některých chůviček je přehrávání ukolébavek. Naše aplikace však nepodporuje ani tuto vlastnost.

Několik chůviček podporuje ukládání přenášeného streamu do souboru. Vznikne tak video, které je možné přehrát v multimediálním přehrávači. Takový soubor může být dobrý např. při zjištění, jak dítě vylezlo z postýlky a do budoucna tomu zabránit. Aplikace též umožňuje ukládání streamu. Ten je ukládán do souboru vždy na straně klienta. Každý připojený klient si může vybrat, zda si bude stream ukládat, či nikoliv.

3 Analýza problému

V této kapitole se budeme zabývat dostupnými technologiemi, jejich popisem a zároveň i výběrem těch nejvhodnějších, které budou dostačující pro naše účely.

3.1 Architektura aplikace

Aplikace, která bude splňovat vlastnosti chůvičky, se musí skládat ze dvou částí. Jedna část bude mít za úkol přijímat obraz a zvuk v pokoji dítěte a druhá musí být schopná dané streamy zobrazit. Mimo to, vysílací část aplikace musí umět obsluhovat více přijímacích zařízení najednou, musí je od sebe jednoznačně rozeznat, umět rozlišit ta, která jsou přihlášená a registrovaná, a musí být schopna držet pro každé zařízení jiný relační klíč určený k šifrování komunikace. Vysílací zařízení tedy poskytuje svým přijímačům určité služby, ke kterým, aby je mohly používat, se musí nejdříve registrovat, a pak přihlásit. Proto byla zvolena architektura klient-server, která odpovídá charakteru chůviček.

Jako server bude dále označován program, jenž je umístěn na zařízení s kamerou a mikrofonem, a má za úkol zaznamenávat zvuk a video a přeposílat ho všem registrovaným klientům, kteří si o něj zažádají (vysílací zařízení). Naopak jako klient bude dále označován program, který je umístěn na zařízení s monitorem a reproduktory, a zobrazuje přijímaný stream od serveru (přijímací zařízení). Aplikací pak rozumíme oba dva programy dohromady, čili celou chůvičku.

3.1.1 Služby serveru

Mezi služby, které bude server poskytovat, se řadí:

- Posílání streamu získaného z kamery a mikrofonu na vyžádání. Posílat je možné buď jen samotné audio nebo audio+video.
- Posílání zpráv při změně videa/audia. Zprávy chodí pouze na vyžádání.
- Registrace nových uživatelů, kteří mohou aplikaci používat.
- Správa serveru přes webové rozhraní.

3.1.2 Možnosti klienta

Klient může využívat všech výše popsaných služeb serveru. Funkce klienta tedy jsou:

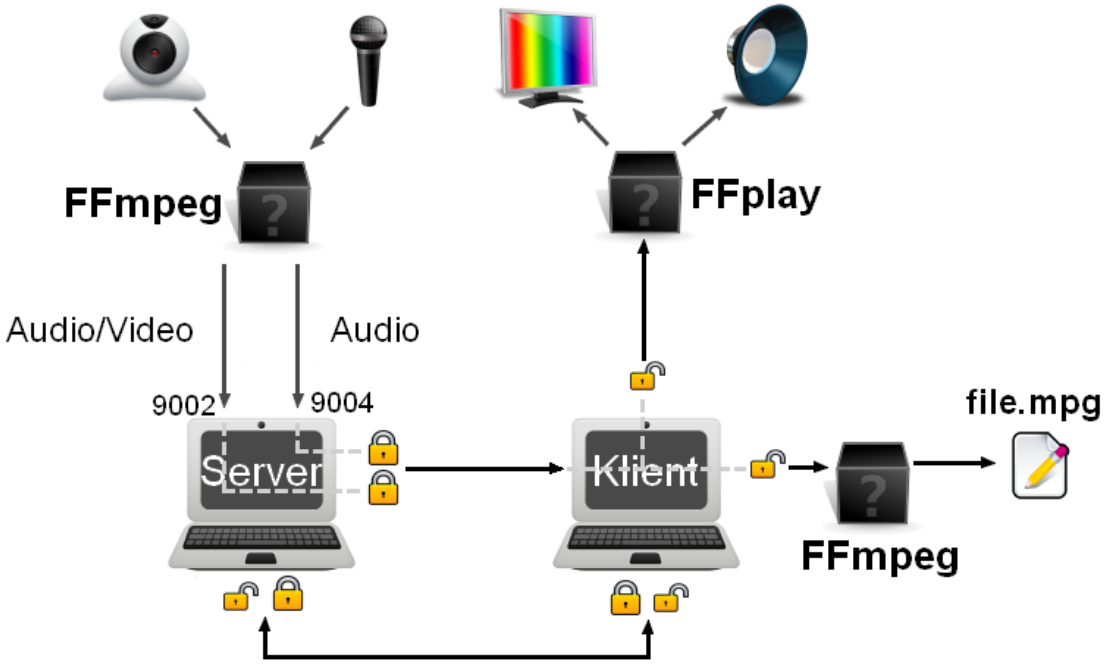
- Vyžádání si a přehrávání audia/videa+audia.
- Zobrazení příchozích zpráv o změně audia/videa.
- Ukládání příchozího streamu do souboru.
- Registrace.

3.1.3 Komunikace

Na schématickém náhledu na obr. 3.1 budou stručně popsány základní prvky komunikace a toku dat v aplikaci. Klient a server jsou zde vyobrazeny jako dva počítače, na kterých aplikace běží. Na serveru běží program *ffmpeg.exe* (viz kap. 3.3.3), který dokáže získávat stream z webové kamery a mikrofonu. Tento stream překóduje na vhodný formát (viz kap. 3.3.2) a na server posílá dva na sobě nezávislé streamy na dva různé porty. První stream je audio+video a druhý je jen audio. Server dále tyto streamy zašifruje a posílá klientovi. Je důležité si uvědomit, že klientovi posílá vždy maximálně jeden stream.

Klient přijímá stream na předem známém portu. Tento stream dešifruje a posílá ho na další port, kde poslouchá program, který je určen pro přehrávání streamu *ffplay.exe* (viz kap. 4.3.5). Kromě zobrazování streamu, může klient posílat dešifrovaný stream ještě na jeden port, na kterém poslouchá *ffmpeg.exe*, který přijatý stream dekóduje a ukládá do souboru.

Kromě této komunikace, kde server posílá stream a která je jednosměrná (stream je posílán pouze ze serveru klientovi), je zde ještě obousměrná komunikace mezi serverem a klientem. Na obrázku je znázorněna nejspodnější čárou, která má na obou koncích šipky. Tato komunikace je též šifrovaná a používá se k výměně komunikačních zpráv mezi servererm a klientem. Tyto zprávy jsou například žádosti o zasílání streamu (klient žádá), přihlášení na server (posílá klient) atd.



Obr. 3.1: Schématický obrázek fungování aplikace.

Celá aplikace byla vytvořena v programovacím jazyce *C++*. Přestože bude aplikace vyvíjena pod systémem Windows 7 s překladačem GCC (MinGW), bude jí možné přeložit i pod systémy GNU/Linux.

3.2 Zabezpečení

Jak bylo uvedeno výše, jeden z hlavních problémů dnešních chůviček je to, že přenos dat mezi přijímacím a vysílacím zařízením není nijak zabezpečený. Kromě toho, že někdo uslyší brečící dítě, jsou zde mnohem závažnější bezpečnostní rizika, která se objevila s nástupem video chůviček. Pokud by se útočníkovi povedlo odchytávat obraz a zvuk z dětské chůvičky, tak se dostává k velice citlivým informacím v podobě vybavení domu, osobních informací členů domácnosti, časových rozvrhů, kdy kdo bude doma atd. Netřeba dodávat, jak by s takovými informacemi bylo možné naložit.

Abychom minimalizovali tato rizika, bude nutné zabezpečit aplikaci na několika úrovních. Prvním krokem by mělo být omezení klientů, kteří se na server mohou

připojit. To lze řešit zavedením registrace klientů viz kap. 3.2.1. Dalším krokem je ověření klientů, kteří se chtějí na server přihlásit a využívat jeho služby. Bude tedy nutné provádět autentizaci klientů viz kap. 3.2.2. Posledním neméně důležitým krokem je samotné šifrování přenášených dat viz kap. 3.2.3 .

3.2.1 Registrace

První krok v sérii zabezpečovacích mechanismů představuje registrace. Odběr streamu klienty ze serveru musí být hlídán, aby nemohlo dojít k tomu, že stream bude odebírat kdokoliv. Bude tedy nutné zavést ověřovací mechanismus (viz kap. 3.2.2), který klienta ověří a pokud bude klienta znát, tak mu zpřístupní dané služby. To znamená, že na straně serveru musí existovat seznam registrovaných klientů, aby bylo proti čemu ověřovat. Možné alternativy, co a proč (ne)ukládat na straně serveru, budou popsány v kapitole 3.2.2. Nyní se budeme zabývat pouze tím, jak jednotlivé klienty na serveru registrovat.

Jednou z možností by bylo, implementovat registraci na straně klienta. To znamená, že klient, kterého by server neznal a který by se tudíž nemohl přihlásit k odběru žádného streamu, by se nejdříve z programu registroval, a pak by se z toho samého programu již mohl přihlásit. To by však umožňovalo registraci komukoliv, a to je právě to, čemu se snažíme vyhnout. Aby bylo možné používat tento mechanismus, museli by klienti při registraci prokazovat, že to jsou legální uživatelé aplikace. To by bylo možné udělat například pomocí certifikátů. Klient by společně s požadavkem registrace posílal i svůj certifikát. Server by si pak mohl kontrolovat, jací klienti žádají o registraci.

Nabízí se i opačná možnost, tedy soustředit veškeré registrace nových klientů na server. Klientská aplikace by pak provedla jen přihlášení k serveru a následně by už mohla přehrávat vyžádaný stream. Pohodlné a přehledné přidávání nových klientů by umožňovalo webové rozhraní serveru. Webové rozhraní by se dalo využít nejen k registraci nových klientů, ale i ke správě serveru a nastavení streamů (viz 3.4). Kvůli širším možnostem využití bylo webové rozhraní zvoleno jako lepší řešení pro registraci klientů.

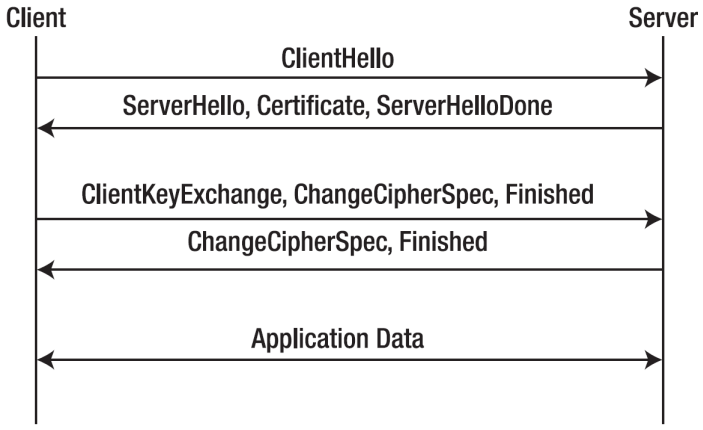
I u tohoto řešení musí dojít k autentizaci uživatele, který se přes webový prohlížeč snaží registrovat nové uživatelské údaje. To je možné udělat tak, že webové rozhraní bude opatřeno heslem. Nikdo jiný, než ten, kdo heslo spolu s aplikací vlastní, nemá

přístup. Vzhledem k přenosu citlivých údajů v podobě nového uživatelského jména a hesla je nutné komunikaci, která probíhá mezi serverem a prohlížečem, šifrovat. Toho lze dosáhnout pomocí protokolu HTTPS.

Protokol HTTPS

Protokol HTTPS (Hypertext Transfer Protocol Secure) je síťový protokol z aplikační vrstvy modelu ISO/OSI. Vychází z protokolu HTTP (Hypertext Transfer Protocol), který je určen pro přenos hypertextových dokumentů. Oproti němu ale navíc dokáže ověřit komunikující protistrany a šifrovat komunikaci pomocí SSL nebo TLS protokolu (dále jen *SSL*). SSL k zajištění bezpečnosti používá jak symetrickou, tak asymetrickou šifru a certifikáty [17]. Než je možné samotné šifrování komunikace, je nejdříve nutné provést takzvaný handshake.

Handshake je množina úkonů, které musí server a klient podstoupit, než si mohou posílat zašifrovaná data. Cílem handshaku je domluva na hlavním sdíleném tajemství, které je použito pro výpočet symetrických klíčů. První posílané zprávy mezi klientem a serverem jsou *hello* zprávy. Díky těmto zprávám si komunikující protistrany vymění informace o tom, kdo používá jakou verzi *SSL*, jaké šifrovací algoritmy atd. V této fázi posílá server svůj certifikát, který si klient ověří. Dále pak, podle typu vybraného šifrovacího algoritmu, dojde k přeposlání počátečního tajemství, které je šifrované veřejným klíčem serveru a používá se ke generování hlavního tajemství. Na konci handshaku jsou z tohoto hlavního tajemství vygenerovány relační klíče pro šifrování komunikace. Proces autentizace serveru pomocí *SSL* je vidět na obr. 3.2. [17]

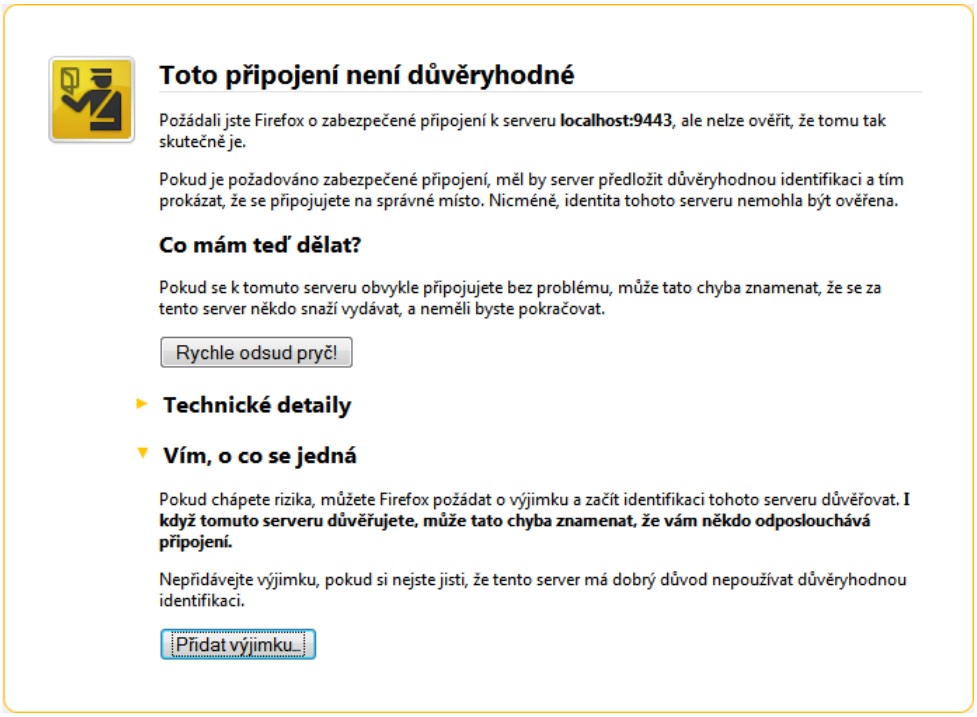


Obr. 3.2: Autentizace serveru pomocí *SSL*. [17]

Certifikát

Aby bylo možné použít HTTPS, bude nutné získat certifikát pro server. Certifikát je dokument digitálně podepsaný certifikační autoritou. Certifikát obsahuje kromě údajů o svém držiteli a o době platnosti i veřejný klíč majitele [17]. Existují dva způsoby, jak získat certifikát. Tím prvním je zažádání o certifikát příslušnou důvěryhodnou certifikační autoritu, která certifikát za poplatek na určitou dobu vydá. Druhá možnost spočívá ve vygenerování takzvaného *self-signed* certifikátu. Jak už název napovídá, tento certifikát není podepsán důvěryhodnou autoritou, ale pouze tím, kdo ho vygeneroval.

Takto podepsaný certifikát samozřejmě není důvěryhodný a je pouze na uživateli, zda ho za něj označí. Nedůvěryhodnost se ve webových prohlížečích projevuje varováním, které je podobné tomu na obr. 3.3.



Obr. 3.3: Varování při použití nedůvěryhodného certifikátu.

Pro naše účely postačí *self-signed* certifikát, jelikož budeme vědět, co schvalujeme. Navíc je i možnost daný certifikát do počítače nainstalovat, a tím se vyhnout varovnému hlášení.

3.2.2 Přihlášení

Dalším krokem v zabezpečení je přihlašování registrovaných uživatelů z klientské aplikace. Aby uživatel mohl sledovat stream, musí nejdříve prokázat, o koho se jedná. Toho uživatel dosáhne zadáním svých přihlašovacích údajů, které se na serveru ověří. Jedná se o údaje, které se ukládají při registraci přes webové rozhraní. Pokud server dané údaje shledá správnými, umožní danému klientovi odebírat stream. Dále si popíšeme možnosti ukládání citlivých údajů na serveru a vybranou metodu pro přihlašovací proces.

Ukládání přihlašovacích údajů na straně serveru

Aby server věděl, komu může a nemůže stream posílat, musí mít list nebo seznam uživatelů s jejich přihlašovacími údaji. V principu existují 3 metody [17], jak lze uživatele autentizovat:

- Řekni, co víš (heslo)
- Ukaž, co máš (platební karta)
- Ukaž, co jsi (otisk prstu)

První metoda je jednoduchá a spočívá v ověření uživatele na základě nějakého tajemství. U této metody se počítá s tím, že tajemství nezná nikdo jiný než uživatel, a tudíž se za něj nemůže nikdo jiný vydávat. Předpokládá se, že síť mezi serverem a klientem je náchylná na odposlechy, a není zde žádná třetí důvěryhodná strana [37]. Výhodou této metody je jednoduchá implementace a nevýhoda je v tom, že lidé si volí příliš krátká a jednoduchá hesla, která nebývá těžké, pomocí slovníkových útoků, prolomit. [17]

Druhá metoda je založená na ověřování podle toho, co uživatel vlastní. Nejčastěji to jsou různé typy karet (OTP, Smart, platební karty). Některé generují nové heslo pokaždé, když se uživatel chce přihlásit (OTP karty), jiné zas umožňují zničení uložených údajů v případě, že se k nim někdo snaží dostat násilnou cestou (Smart karty). Tento způsob ověřování není pro naši aplikaci příliš vhodný. [17]

Posledním a nejsložitějším typem, co se implementace týče, je ověřování uživatele na základě biometrických údajů. Používá otisk prstu nebo sken rohovky. Možné je i užití podpisu uživatele, kde se monitoruje tlak vyvíjený při psaní a časování jednotlivých smyček. Nevýhodou bývá časté odmítnutí pravého uživatele a přijetí útočníka. Nevýhoda u biometrické autentizace je správa klíčů. Ty jsou generovány z údajů uživatele (otisku prstu, skenu rohovky, atd.) a jsou pro daného uživatel unikátní. Pokud někdo toto ukradne, zkopíruje nebo jinak získá, původní uživatel si nemůže nechat vygenerovat novou rohovku, jako by to bylo možné v případě ztráty hesla. [17]

Z výše uvedených metod se pro naše účely jako nejlepší jeví metoda první, tedy ověřování uživatele na základě toho, co zná. Tato metoda má několik úrovní zabezpečení. Tou první, dalo by se říci nulovou, je uložení páru *uživatelské jméno - heslo* na straně serveru v prostém textu. To představuje obrovské bezpečností riziko. V případě, že by se na server někdo dostal a ukradl takový seznam uživatelů, má přímý přístup ke všem uživatelským účtům.

Lepší řešení představuje použití hashovací funkce, která se použije na hesla. Na straně serveru pak nejsou uložena hesla, ale jen jejich hashe. Hash je jednocestná funkce, tedy ze zahashovaných hesel bychom neměli být schopni získat původní text. O tom, co je hashování a jaké by se měly používat hashovací funkce, si povíme v kap. 3.2.2. Když uživatel zadá své přihlašovací údaje, tak se z jeho hesla vytvoří hash a porovná s tím, jenž je uložen v databázi serveru. Pokud je stejný, uživatel je autentizován.

Pokud bychom chtěli útočníkovi napadení ztížit, je možné použít takzvané solení (*salting*). Solení je proces, kdy se do hashovaného hesla přidává další informace [17]. Tato informace se nazývá sůl (*salt*) a jedná se o náhodné číslo, které musí být uloženo spolu s hashem hesla na straně serveru. Aby útočník odhalil heslo při použití hashů, musel by vytvořit slovník, ve kterém by byly všechny možné kombinace znaků. Každou jednotlivou kombinaci by musel hashovat a porovnávat se získaným heslem. Při velikosti slovníku n slov by útočník musel provést n hashování. Pokud se při hashování hesla použije ještě sůl, která je dlouhá k bitů, pak už útočník musí provést $2^k n$ hashů [17]. Omezení této metody je v tom, že není stavěná na offline slovníkové útoky [17].

Poslední úrovní zabezpečení ukládaných údajů by bylo použití šifrování. Jak uživatelská jména, tak hesla v jakémkoliv formátu, by byla šifrována použitím například symetrické šifry.

Vzhledem k tomu, že počet potenciálních uživatelů, kteří by se na server připojovali, je velice malý (do 10), nebyla pro ukládání uživatelských údajů zvolena databáze, ale pouze soubor. Databáze by nenašla uplatnění ani v jiné části aplikace. Podle výše uvedených úrovní ukládání hesla, byly zvoleny poslední dvě, tzn. do souboru budou ukládány uživatelská jména spolu s hashem a solí a soubor bude šifrován symetrickou šifrou AES (viz kap. 3.2.3), kde klíč pro tuto šifru je nadefinován jako proměnná.

Autentizační protokol [37]

Jedna věc je, jak uložit a porovnávat údaje od klienta na straně serveru, a druhá, jak je tam dostat. Za tímto účelem vznikl již v roce 1997 protokol SRP (Secure remote password ptotocol), který slouží k autentizaci pomocí hesla a k výměně klíčů. Tento protokol patří do skupiny AKE (Asymmetric key exchange) a jeho hlavní funkcí je výměna klíčů a jejich využití k ověření, že obě strany znají heslo. Na rozdíl od EKE (Encrypted key exchange) nešifruje přenášené zprávy, ale místo toho používá předdefinované matematické vztahy ke sloučení náhodných hodnot a hesla. Další rozdíl od EKE představuje to, že se neposílá přímo heslo, ale pouze jeho hash (v původním textu označovaný jako *verifier*). Klientovo heslo tak nikdy neopustí systém a nedostane se do sítě. Samotný hash hesla však k imitování uživatele nestačí, původní heslo je stále třeba.

Veškeré početní operace jsou prováděny v konečné množině $GF(n)$. Tzn. velké prvočíslo n je vybráno předem a veškeré sčítání, násobení a umocňování je modifikováno modulo n. Všechny vstupní i výstupní parametry jsou tak čísla na intervalu *<0, n-1>*. Dále bude na příkladu popsán autentizační proces.

Řekněme, že máme dvě osoby, Alici a Boba, a budeme chtít Alici přihlásit u Boba. Nejdříve musíme k Bobovi uložit hash Alicina hesla (*verifier*) a náhodně vybranou sůl. Hash v se spočítá jako:

$$x = H(s, P)$$

$$v = g^x$$

x je zahozeno, jelikož odpovídá otevřené formě hesla P. U AKE protokolů se obvykle používá i heslo a veřejný klíč na straně Boba, který by schraňovala Alice. U protokolu SRP dojde ke vzájemnému ověření už jen tím, že Bob bezpečně uchovává

Alicin *verifier*. To zjednodušuje celý protokol, protože Alice si nemusí pamatovat Bobův klíč. V tabulce 3.1 je uveden postup vzájemné autentizace Boba a Alice krok za krokem.

	Alice		Bob
1.		$\xrightarrow{C}$	(lookup s,v)
2.	$x = H(s,\ P)$	$\xleftarrow{s}$	
3.	$A = g^a$	$\xrightarrow{A}$	
4.		$\xleftarrow{B,\ u}$	$B = v + g^b$
5.	$S = (B - g^x)^{a+ux}$		$S = (Av^u)^b$
6.	$K = H(S)$		$K = H(S)$
7.	$M_1 = (A,\ B,\ K)$	$\xrightarrow{M_1}$	ověření M_1
8.	ověření M_2	$\xleftarrow{M_2}$	$M_2 = H(A,\ M_1,\ K)$

Tab. 3.1: Postup autentizace SRP protokolu [37].

1. Alice pošle Bobovi svoje uživatelské jméno (např. alice).
2. Bob vyhledá vstupní údaje o Alici a stáhne si její *verifier* a sůl. Sůl s pošle Alici a ta si z její pomocí a skutečného hesla spočítá privátní klíč x.
3. Alice náhodně vygeneruje číslo a, pro nějž platí: $1 < a < n$. Pak spočítá dočasný veřejný klíč $A = g^a$ a pošle ho Bobovi.
4. Bob vygeneruje náhodné číslo b, kde platí $1 < b < n$ a spočítá svůj dočasný veřejný klíč $B = v + g^b$. Tento klíč pošle zpět Alici spolu s náhodně vygenerovaným parametrem u.
5. Oba si nyní spočítají $S = g^{ab+bux}$ pomocí hodnot, které má každý dostupné. Pokud Alicino heslo, které zadala v kroku 2, je shodné s originálním heslem, které se použilo na generování hashe, pak se obě hodnoty S shodují.
6. Obě strany si pomocí S spočítají relační klíč (*session*).
7. Alice pošle Bobovi M_1 jako důkaz, že má správný relační klíč. Bob si sám spočítá M_1 a porovná je.
8. Bob pošle Alici M_2 jako důkaz, že má správný relační klíč. Alice také ověří M_2 a přijme ho jen, pokud se shoduje s jejím výpočtem.

Výsledný relační klíč je při každém přihlášení jiný. Další podrobnosti, jako např. jakou roli hraje parametr u, se lze dočíst v [37].

V roce 2002 vyšla nová verze protokolu SRP-6. V této verzi protokolu, který odolává jak aktivním, tak pasivním síťovým útokům i přes použití krátkých dobře zapamatovatelných hesel, byly odstraněny některé menší bezpečností nedostatky, která původní verze skrývala. Jeden takový nedostatek útočníkům umožňoval uskutečnit a ověřit 2 pokusy o uhádnutí hesla při napodobení přihlášení [36]. Tato vlastnost nepředstavuje závažnou bezpečností hrozbu, jelikož každý pokus o uhádnutí má za následek detekovatelné selhání na obou stranách a stále by bylo potřeba nereálného počtu on-line pokusů, a to i v případě, že by se toto číslo snížilo na polovinu [36]. Jednoduchou změnou protokolu popsanou v [36] se útoku *Two-for-one*, jak je v originálu nazýván, dá zabránit.

Tento protokol se jeví jako velice účinný nástroj určený pro bezpečnou autentizaci bez možnosti napadaní útočníkem nebo odposlechu hesla. Jedinou vadu představuje to, že již počítá s tím, že na straně serveru jsou přihlašovací údaje uloženy, a nezabývá se jejich bezpečným přenesením od klienta na server. Naštěstí pro nás je tento problém je už vyřešen v kapitole 3.2.1, a tak je možné protokol SRP-6 bez problému použít.

Hashovací funkce

Hashovací funkce mapuje dlouhý řetězec na kratší a běžně se používá v hashovacích tabulkách, kde má za úkol urychlit přístup k jednotlivým položkám. Bezpečná šifrovací funkce má, oproti té běžně používané, jejímž hlavním úkolem je rovnoměrné rozdělení dat, několik vlastností navíc [17]:

- Neměla by spotřebovávat příliš mnoho výpočetního času pro výpočet hashe ani pro relativně dlouhé řetězce - efektivnost.
- Z hashe by nemělo být výpočetně možné získat původní řetězec - jednocestná funkce.
- Pro každé dvě rozdílné zprávy neexistuje stejný hash - bezkolizní funkce.

Dříve se používala hashovací funkce MD5. Tato funkce je však náchylná na kolizní útoky. První takový byl popsán již v roce 2004 (viz [34]). Od té doby došlo k

vylepší MD5, ale zároveň i přibyly další algoritmy (viz [27]) na vyhledávání kolizí a MD5 se již nepovažuje za bezpečnou hashovací funkci.

Místo MD5 se dnes používá SHA-256 nebo SHA-512. Které vychází ze svého předchůdce, SHA-1, ale produkují delší výstupní hashe (256 a 512 bitů). [17]

Multicast [16]

Aby se klient dokázal připojit na server, tak musí znát IP adresu serveru a jeho port. Pro snadnější použití je možnost implementovat vyhledávání serveru pomocí doménového jména. Jména si lidé zapamatují snáze než IP adresy. Toho je možné dosáhnout pomocí služby, která se jmenuje Multicast DNS (dále mDNS).

mDNS se používá k vytvoření DNS záznamů v síti (privátní), kde žádný DNS server není. Používá k tomu dobře známé funkce a operace z DNS (Domain Name System) rozhraní. Místo unicastu se však používá multicast. Každý počítač v síti má svojí tabulku záznamů a musí být připojen do mDNS skupiny. Tato skupina používá IP adresu *224.0.0.251* a port *5353*. Pokud potřebuje klient znát IP adresu stroje, pošle této skupině zprávu se jménem. Zpátky mu odpoví pouze ten, kdo toto jméno zná, a ve zprávě pošle i příslušnou IP adresu.

Formát jmen, který se pro mDNS používá, je ve tvaru *vlastni-nazev-domeny.local.* Kde sufix *.local.* určuje lokalizaci jména na lokální síť.

3.2.3 Komunikace

Posledním krokem v sérii zabezpečení je šifrování navázaného spojení. Nejdříve ale musíme určit, jaké typy komunikace budou v naší aplikaci probíhat. Tradiční pojetí architektury klient - server, kdy server pouze poslouchá, a až k němu dorazí zpráva, tak na ní odpoví, je pro naše účely nedostačující. Je to z toho důvodu, že i server bude potřebovat posílat zprávy nezávisle na tom, jestli zrovna někomu odpovídá či nikoliv. Mezi tyto zprávy patří upozornění, zda se změnil obraz nebo hlasitost. Komunikace mezi serverem a klientem bude tedy obousměrná ve smyslu, že jak jeden, tak druhý budou moct poslat zprávu kdykoliv. S tím, že na zprávy od klienta bude server odpovídat, ale obráceně nikoliv.

Vedle této obousměrné komunikační linky bude mezi serverem a klientem existovat ještě jedna jednocestná linka. Touto linkou se budou posílat multimediální data ze serveru klientovi. Klient bude tato data jen přijímat a žádným způsobem je nebude potvrzovat.

Na realizaci těchto komunikačních linek máme 2 možnosti, co se týče použití transportních protokolů (TCP, UDP).

Komunikační protokoly

Protokol TCP (Transmision Control Protocol) poskytuje spojované služby. To znamená, že mezi oběma konci spojení vytváří na čas virtuální okruh, který je plně duplexní. Pokud se data ztratí nebo poškodí, jsou vyžádána znovu a jejich integrita je zabezpečena kontrolním součtem. Aplikace se tedy již nemusí starat o to, jestli se nějaká data ztratila. Protokol TCP neobsahuje žádnou ochranu proti modifikacím zpráv a jejich kontrolních součtů. [9]

Aplikace využívající TCP protokol pro komunikaci má tu vlastnost, že pokud se spojení přeruší, tak funkce, která slouží pro příjem dat (např. recv, read...), skončí chybou. Tímto se dá detekovat, že se klient/server odpojil nebo nastalo přerušení spojení. Této vlastnosti a dalších jmenovaných, které protokol TCP poskytuje, lze využít při vytváření prvního komunikačního kanálu, který slouží pro přenos zpráv mezi serverem a klientem. V případě, že se klient neočekávaně odpojí, server to zjistí a může se podle toho zachovat (vymazání klientových údajů jako jeho id nebo uloženého nastavení ohledně zasílání zpráv o detekci změn atd.). To samé platí i pro klienta, který díky tomu nebude do nekonečna čekat na odpověď serveru. V případě, že by TCP protokol touto vlastností nedisponoval, pak by bylo nutné implementovat cyklické dotazování, jestli aplikace ještě žije.

Protokol UDP (User Datagram Protocol) poskytuje, na rozdíl od TCP, nespojované služby. To znamená, že nenavazuje spojení a nestará se o to, jestli odeslaný datagram příjemci dojde nebo ne [9]. UDP protokol je možné použít pro přenos streamu, jednak proto, že pokud se ztratí jeden datagram s video rámcem, tak na výsledném obrazu to většinou není pozorovatelné, a také proto, že právě neobsahuje potvrzovací mechanismus. Video rámec, který obsahuje jeden obrázek a který přijde

mimo své pořadí (pozdě) je stejně zahozen, a proto nemá smysl ho za každou cenu vyžadovat.

Vzhledem k tomu, že ze serveru je potřeba přenášet stream klientům, tak i protokol UDP zde najde své uplatnění. Další možnosti přenášení streamů budou popsány v kap. 3.3.4.

Nyní se dostáváme k možnostem šifrování přenosu. Popíšeme několik blokových šifer, které je možné pro šifrování komunikace použít. Pro šifrování obou kanálů bude vybrána stejná šifra. Vzhledem k velkému množství dat, která se přenáší po streamovacím kanálu, je lepší volbou symetrická šifra. Asymetrické šifry se v praxi nedají použít pro velké množství dat, neboť jsou příliš pomalé.

DES [17]

První šifra, kterou zmíníme, je šifra DES (Data Encryption Standard). Data jsou šifrována po 64 bitových blocích pomocí 64 bitového klíče. Respektive se používá 56 bitový klíč, jelikož 8 bitů se používá pro detekci poškození klíče. Dřív se DES používal ve všech možných odvětvích včetně finančního. Pak se ji ale podařilo prolomit pomocí hrubé síly, kde útočník na zašifrovaný text zkoušel jeden klíč za druhým (existuje "jen"c^{56} možností).

Po prolomení šifry DES byla vytvořena nová šifra, Triple DES. Jak už název napovídá, jedná se o bezpečnější verzi původní šifry DES. U této šifry dochází k zašifrování dat třikrát a pokaždé lze použít jiný klíč. Zpráva se nejdříve zašifruje prvním klíčem, pak se druhým provede dešifrování a třetím se opět zašifruje. Tento zvláštní postup se dělá kvůli zpětné kompatibilitě a jednodušší možnosti přechodu z DES na Triple DES. Oproti původní verzi je tato prakticky třikrát pomalejší a i od jejího používání se ustupuje.

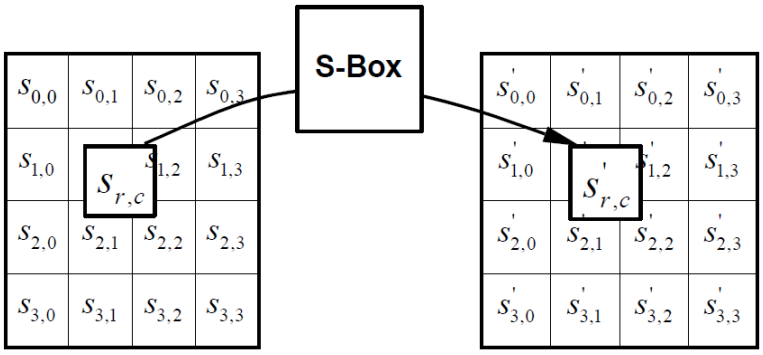
AES

Šifra AES (Advance Encryption Standard) byla vytvořena jako náhrada za prolomenou šifru DES. Oproti dvěma výše zmiňovaným je bezpečnější a rychlejší. Zatím nebyla prolomena a uvádí se, že super počítač s výpočetním výkonem 10,51 * 10^{15}

FLOPS[1] by pomocí hrubé síly potřeboval 1,02 * 10^{18} let na prolomení (na DES by mu stačilo 399 sekund) [5]. AES je symetrická šifra, která šifruje data po pevně daných délkách, a taktéž pracuje s délkami klíčů 128, 192, 256 bitů [17]. Jiné délky standard nedovoluje.

AES vnitřně pracuje s maticí, která se nazývá stav (*state*). Délka vstupního bloku, výstupního bloku i stavu je 128 bitů. Stav je pak tvořen maticí o rozměrech 4x4. Na začátku šifrování jsou vstupní byty nakopírovány do stavové matice. Následně jsou na ni aplikovány šifrovací operace a výsledek je pak zkopírován na výstup. Počet cyklů, které se vykonají během algoritmu, je dán délkou klíče, 128b = 10 kol, 192b = 12 a 256b = 14 kol. Během každého kola se vykonají následující operace [4]:

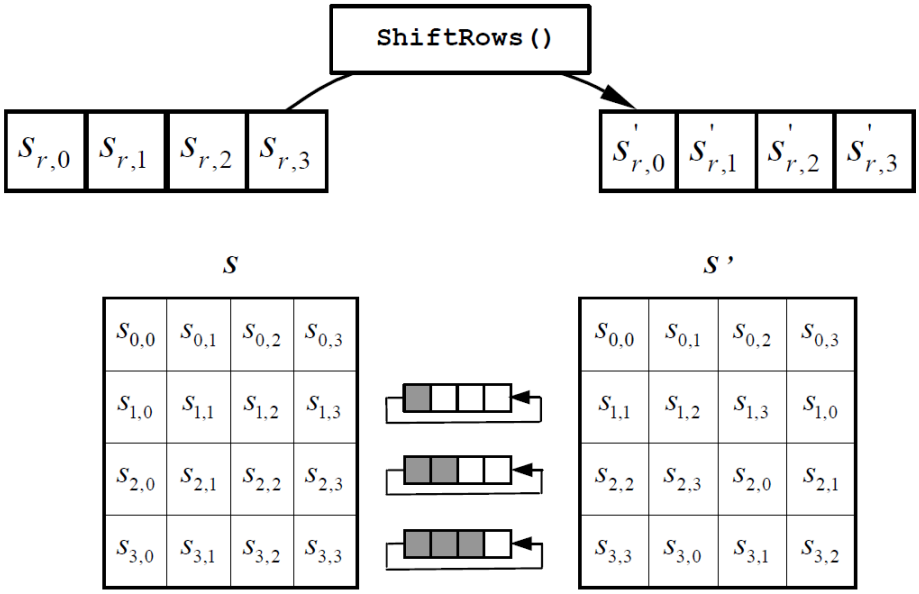
- *SubBytes* - Jedná se o nelineární substituci bytů, která se aplikuje na každý byte zvlášť za použití substituční tabulky (*S-box*) (viz obr. 3.4).
- *ShiftRow* - Spodní 3 řádky stavové matice jsou posunovány o určitý díl doleva.
- *MixColumn* - Lineární transformace sloupečků stavu.
- *AddRoundKey* - Každý byte stavu je sloučen s klíčem cyklu, který je každé kolo jiný a který se derivuje z šifrovacího klíče.



Obr. 3.4: Aplikace *S-box* na každý byte u šifry AES v *SubBytes* fázi [28].

V posledním kole se *MixColumn* vynechává. Pseudokód k průběhu šifry je na obr. 3.6 [4].

[1]Floating-point Operations per Second - počet operací v plovoucí řádové čárce za sekundu



Obr. 3.5: Posun řádků u šifry AES v *ShiftRow* fázi [26].

Míra bezpečnosti AES závisí nejen na tom, že šifrovací klíč musí být dostatečně náhodný, ale i na tom, jaký šifrovací mód se použije. Módy, které lze použít, jsou ECB, CBC, CFB a OFB [10].

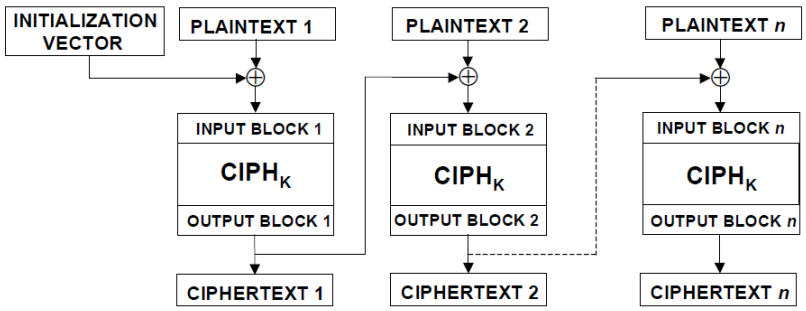
ECB (Electronic Codebook Mode) mód aplikuje šifru na bloky dat tak, jak jdou za sebou. V tomto módu je možné (de)šifrovat více bloků paralelně. Problém ale představuje to, že určitý blok textu bude šifrován pokaždé stejně. To může způsobovat opakující se sekvence a určitou čitelnost kódu, jako je vidět na obr. 3.9 (b).

Bezpečnější metoda je CBC (Cipher Block Chaining Mode), která nejdříve text XORuje s předchozím zašifrovaným blokem, a až pak ho také zašifruje. Díky tomu jsou na sobě jednotlivé bloky závislé. Pro úplně první blok ještě neexistuje blok předchozí, a tak se používá inicializační vektor (IV). Tento vektor nemusí být tajný, ale musí být nepředvídatelný. Postup šifrování je ukázán na obr. 3.7. Vzhledem k tomu, že bloky jsou na sobě závislé, není možné jejich paralelní (de)šifrování [10]. Metoda CBC je oproti ECB mnohem bezpečnější, což je vidět i na zašifrování obrázku 3.9 (c). [10]

Metoda CFB (Cipher Feedback Mode) je podobná metodě CBC. K šifrování bloku také využívá blok předchozí. Metoda pracuje tak, že na začátku se vezme

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

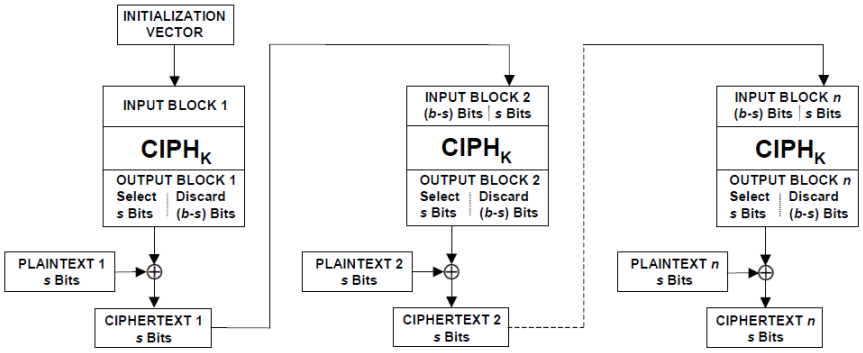
Obr. 3.6: Pseudokód šifry AES [23] (obr. byl upraven).



Obr. 3.7: Šifrování AES pomocí metody CBC [30] (obr. byl upraven).

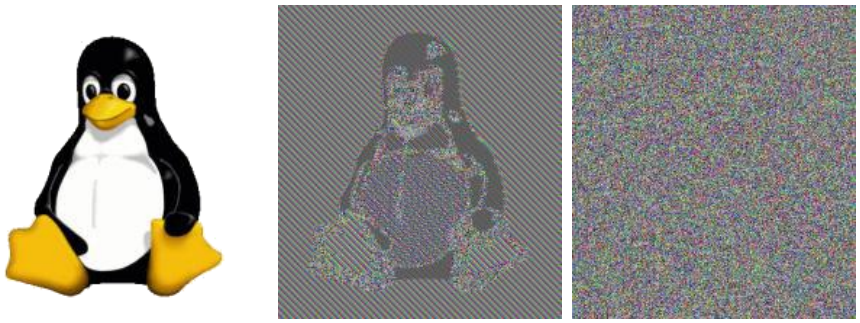
inicializační vektor a ten se zašifruje. Z tohoto zašifrovaného bloku se vezme s nejvýznamnějších bitů a ty jsou XORovány s originálním textem k zašifrování, tím vznikne první zašifrovaný segment. Z IV se následně vezme b-s nejméně významných bitů, spojí se s s prvního zašifrovaného segmentu a vytvoří tak další vstupní blok. Stejně jako u CBC metody, ani zde není možné použít paralelní šifrování, jelikož bloky na sobě závisí. [10]

Metoda OFB (Output Feedback Mode) používá následující postup při šifrování: vektor IV je zašifrován a vznikne první výstupní blok, který je XORován s textem k zašifrování, a tím vznikne první zašifrovaný blok. Šifra je dále aplikována na první výstupní blok, a tím vznikne druhý výstupní blok.



Obr. 3.8: Šifrování AES pomocí metody CFB [31] (obr. byl upraven).

Druhý výstupní blok je XORován s druhým textem k zašifrování atd. Vektor IV musí být jedinečný pro každý běh (každou zprávu) s daným klíčem. [10]



(a) Originální obrázek (b) Metoda ECB (c) Metoda CBC

Obr. 3.9: Ukázka použití ECB a CBC módů na obrázek [7].

Původním záměrem bylo použití klíče o délce 512 bitů kvůli vyšší bezpečnosti. Standard AES však stále umožňuje použití maximálně 256 bitového klíče. Nakonec byl tedy zvolen klíč o délce 256 bitů a metoda CBC, kterou budou šifrována data v obou komunikačních kanálech. Problém s distribucí symetrických klíčů zde odpadá, jelikož výsledkem autentizačního procesu protokolu SRP je relační klíč, který je možné dále použít k šifrování pomocí AES (viz kap. 4.2.2).

3.3 Stream

Jak už bylo zmíněno výše, server musí umět poskytnout dva streamy k odběru. Tím prvním je audio+video stream, a tím druhým pak pouze audio stream. V této kapitole si přiblížíme stream, jeho možnosti, co se týče formátů audia/videa, a také se podíváme na způsoby, jak je možné dopravit stream ze serveru klientovi. Důležitou část tvoří i kapitola o možnostech detekce změny obrazu nebo hlasitosti.

Streamování je technika používající se především pro přenos multimediální informace přes internet. Při streamování je možné daný video soubor spustit již během stahování. Což je výhodné, ale na druhou stranu to klade větší nároky na klienta, který stream odebírá. Typy streamu jsou obecně dva, první je *on demand*, tedy stream na vyžádání (např. Youtube), a tím druhým jsou přenosy v reálném čase, které se budou týkat i naší aplikace.

Datové prvky obsažené ve streamu se označují jako rámce (*frame*). Každý rámec je dekódován jiným typem CODECu, podle toho, o jaký typ rámce se jedná (audio, video). CODEC (coder-decoder) určuje, jakým způsobem jsou data komprimována a uložena. Komprimovaná data neobsahují všechny informace jako data původní, kapacitně zabírají mnohem méně místa a jsou díky tomu vhodnější pro přenos po síti.

3.3.1 Audio

Audiem budeme dále označovat digitální audio, které se dá získat pomocí A/D převodu. A/D převodem rozumíme převod analogového signálu na digitální. Audio se skládá z velkého počtu vzorků, které představují hodnotu původního signálu v určitém čase. Audia jsou nahrávána s různou vzorkovací frekvencí, která udává, jak rychle se má každý vzorek přehrávat, a je dána počtem vzorků za sekundu. Nejčastěji používané frekvence jsou 44 100Hz a 22050Hz.

Intenzitu zvuku je možné určit pomocí decibelů - *dB*. Jedná se o veličinu, která je vyjádřena jako logaritmus podílu dvou hodnot. Padesáti decibely se dá označit běžná konverzace a 120 decibelů představuje práh bolesti. [33]

MP3

Pro přenos audia byla zvolena komprimace pomocí MP3 (MPEG Layer III Audio Encoding). MP3 komprese používá percepční kódování (metoda založená na vnímání zvuku člověkem - *psychoacoustic*). Tato metoda dovoluje odstranit nebo redukovat přesnost takových částí audia, které jsou pro lidský sluch méně slyšitelné. MP3 soubor, jehož přenosová rychlost je 128 *kbit/s*, tvoří asi 1/11 původního nekomprimovaného LPCM (Linear pulse-code modulation) souboru, který má kvalitu kompaktního disku (44 100 Hz, 16 bitů hloubku). [20]

3.3.2 Video

Jako digitální video se dá označit sekvence rámců, kde každý rámec se skládá z obdélníkové mřížky obrazových elementů - pixelů. Každý pixel může být buď jeden bit (pouze černobílý obraz) nebo 8 bitů (256 odstínů šedi). Pro dosažení barevného obrazu se většinou používá 8 bitů pro každou barevnou složku z RGB (red - green - blue). To představuje 24 bitů na pixel, a tedy okolo 16 milionů barev (více, než lidské oko může rozlišit). Obvyklé snímkovací frekvence (*frame rate*) se pohybují od 24 do 30 *snímků/s*. [33]

Přenos videa po síti je náročný na přenosovou kapacitu, není-li použita komprese. Pro představu[2]:

- Video, které má velikost 640x480, 24 bitů barevné informace na pixel a 30 *snímků/s*, potřebuje ke svému přenosu 421 *Mb/s* [33].
- Video velikosti 720x576, s 8 bitovou barevnou hloubkou a 25 *snímky/s*, potřebuje šířku pásma 158 *Mb/s* [19].
- HDTV (High Definition Television) už potřebuje 1,85 *Gb/s* [19].

MPEG

MPEG standard (Moving Picture Coding Exports Group) patří mezi nejznámější a asi i nejpoužívanější kódování videa (umí komprimovat ale i audio [33]).

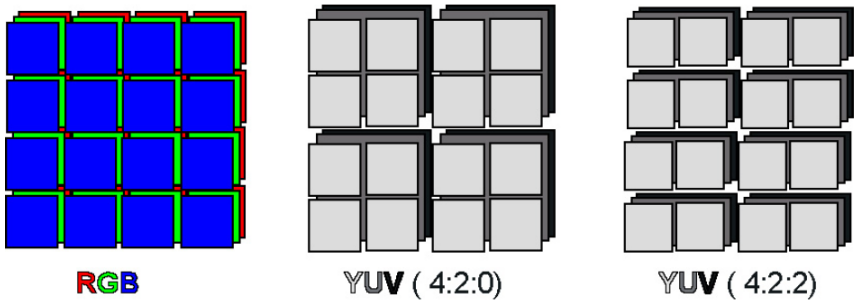
[2]Všechny výpočty počítají s barevným modelem YUV 4:2:2 a byly ověřeny pomocí [13].

Tento standard zahrnuje kompresní metody: MPEG-1, MPEG-2, MPEG-4. MPEG-4 je nejmladší a byl konstruován na použití v multimediálních aplikacích a pro video komunikace [19].

Základním stavebním kamenem MPEG streamu jsou komponenty nazývající se elementární stream (Elementary Stream) (dále ES). Každé video může obsahovat několik typů ES: audio, video, titulky atd. Jednotlivé ES jsou ukládány do větších celků, kterým se říká PES pakety (Packetised Elementary Stream). Každý PES paket má 6 bytovou hlavičku, která mimo jiné udává i typ přenášených dat (stream ID je ve čtvrtém bytu hlavičky a má tvar: 110x xxxx - audio, 1110 yyyy - video). [11]

MPEG komprimuje data v 5 krocích: redukce rozlišení, nahrazení pohybu, diskrétní kosinová transformace (DCT), kvantizace a kódování entropie [19]. Dále si blíže přiblížíme první dva kroky.

Redukce rozlišení využívá toho, že lidské oko je méně citlivé na barevnou informaci než na kontrasty tmavé a světlé a že provádí převod z prostoru RGB na YUV komponenty. Barevné složky U a V mohou být redukovány (převzorkovány) na polovinu pixelů v horizontální úrovni (YUV 4:2:2) nebo zároveň i ve vertikální (YUV 4:2:0). Toto převzorkování má za následek snížení objemu dat o 50% a 33%. [19]

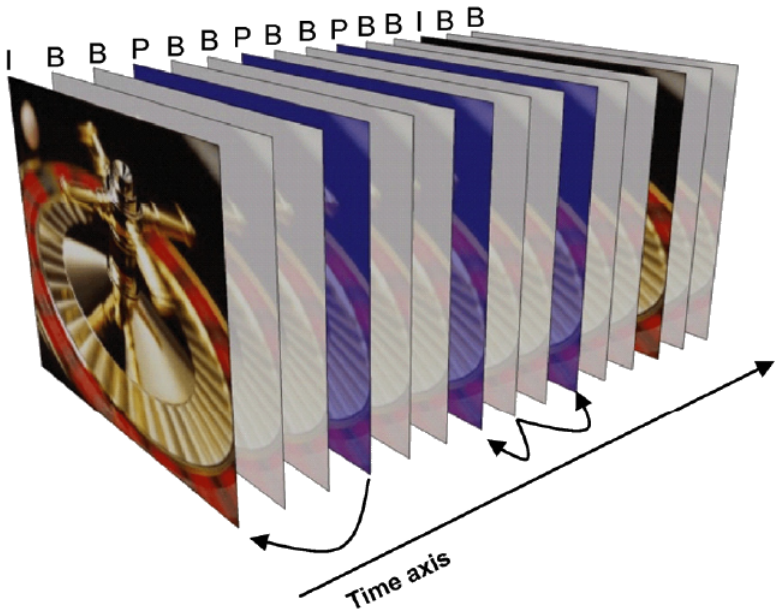


Obr. 3.10: Spojení 2(4) pixelů chrominančního kanálu dohromady [8].

V druhém kroku, nahrazení pohybu, se využívá redundance, která se ve videu objevuje. Tato redundance spočívá v tom, že dva po sobě jdoucí rámce, jsou často téměř identické. To platí hlavně pro scény, kde se kamera nehýbe, a osoby se pohybují velice pomalu. MPEG tak rozlišuje 3 typy rámců [33]:

- *I* rámce (Intracoded): obsahují celý obrázek.
- *P* rámce (Predictive): obsahují rozdíl oproti předchozímu rámci (*P* nebo *I*). Bez referenčního rámce je nejde zrekonstruovat.
- *B* rámce (Bidirectional): obsahují rozdíl, jako *P* rámce, ale na obě strany (vzhledem k předchozímu a následujícímu rámci).

Podle počtu použitých typů rámců se odvíjí i výsledná kvalita a velikost videa. Čím více *I* rámců, tím větší kvalita a velikost, a naopak, čím více *B* rámců, tím je kvalita horší a velikost menší [19]. Možný sled rámců videa je vidět na obr. 3.11. *B* rámce se, kvůli svým nárokům na buffer a větší složitosti, ne vždy používají [33]. Vztah (pohybový) mezi dvěma rámci se určuje pomocí pohybového vektoru. Jaké jsou výpočetní možnosti toho algoritmu se lze dočíst v [3], [19] a [33].



Obr. 3.11: Ukázka posloupnosti rámců v MPEGu [2].

MPEG-TS [11]

MPEG-TS (MPEG transport stream) je kontejner používající se na přenos dat (audio, video atd.) po nespolehlivé síti. Oproti programovému streamu obsahuje robustní mechanismus na opravu chyb a synchronizaci. Transportní stream se skládá

z TS paketů, které mají pevně danou délku - $188B$ ($4B$ tvoří hlavička). Jeden PES paket bývá rozdělen do několika TS paketů a nachází se v části pro data. V hlavičce TS paketu je pak jeden bit nastaven na indikaci začátku PES paketu. Vzhledem k tomu, že PES paket může mít různou délku, musí se případné zbývající místo v TS paketu doplnit *0xFF*, aby další PES paket začínal na začátku nového TS paketu. V každém TS paketu smí být začátek jen jednoho PES paketu. Vzhledem ke těmto vlastnostem byl tento kontejner vybrán pro přenos obou streamů, které se v aplikaci přenášejí.

3.3.3 Knihovna pro práci s audiem/videem

Pro získávání audia a videa ze vstupních zařízení je nutné vybrat vhodný nástroj a dále síťové protokoly, pomocí kterých bude stream dopraven k příjemci (viz kap. 3.3.4). Pro získání i posílání audia/videa se nabízí použití již existujících nástrojů jako je např. Skype. Tyto nástroje však nejsou dostupné na úrovni zdrojového kódu, a navíc u nich není možnost dostat se k posílaným datům, aby je bylo možné zkoumat kvůli detekci změn. Proto byla vybrána jiná varianta v podobě multiplatformní opensourcové knihovny - FFmpeg.

FFmpeg představuje kompletní řešení pro nahrávání, převod a streamování audia/videa [12]. FFmpeg poskytuje knihovny napsané v programovacím jazyce *C*, který umí provádět výše zmíněné (např. avcodec, avdevice, avutil atd.) nebo poskytuje již hotové nástroje (přeložené nebo zdrojové kódy) na též výše zmíněné aktivity (*ffmpeg.exe, ffprobe.exe, ffserver.exe, ffplay.exe*).

- *ffmpeg.exe* - Nástroj pro konverzi audia/videa, který dokáže získávat vstup(y) z připojených zařízení (soubor, internetový stream, pipe, multimediální zařízení) a posílat je na výstupy (soubor, internet atd.).
- *ffprobe.exe* - Slouží k získávání informací o daném streamu.
- *ffserver.exe* - Jedná se o streamovací server, který podporuje audio i video.
- *ffplay.exe* - Nástroj určený pro přehrávání multimédií buď ze souboru nebo internetu. Používá knihovnu SDL.

Máme tedy dvě možnosti, jak FFmpeg použít. Jedna je rovnou pomocí knihoven a druhá pomocí již vytvořených nástrojů. FFmpeg poskytuje rozsáhlou, čitelnou a

srozumitelnou dokumentaci k tomu, jak používat výše zmíněné nástroje. Popisuje zde, jaké parametry se na co používají, jaké mají přípustné hodnoty atd. Dokumentace, kde by bylo popsáno, jak používat knihovny a jejich funkce ve zdrojovém kódu, až na cca 6 vzorových příkladů poskytovaných FFmpegem, neexistuje. Další nevýhoda použití knihoven je ta, že jednotlivé updaty nejsou zpětně kompatibilní. Pokud se tedy vydá update, pak se musí celý napsaný kód projít a opravit. V některých případech jde o malé změny typu: přidání za název funkce o jedničku větší číslo. Pokud jsou ale změny velké, může jít o změnu celé logiky prováděného úkonu, a pak jsou zásahy do kódu mnohem větší. Přesně takové jsou rozdíly mezi verzemi 1.2 a 2.0 knihovny SDL (Simple DirectMedia Layer). Tato knihovna byla uvažována pro přehrávání doručeného streamu a nástroj *ffplay.exe* ji k tomuto účelu též používá. Knihovna SDL verze 2.0 je na tom s dokumentací podobně jako FFmpeg pro své knihovny.

Po zhodnocení výše uvedených důvodů, bylo jako lepší řešení vybráno použití již hotových nástrojů, konkrétně pak *ffmpeg.exe* pro získání vstupu z kamery a mikrofonu a *ffplay.exe* pro přehrávání streamu na straně klienta. Tato varianta umožňuje jednoduché vyměnění nástrojů v případě updatu bez zásahu do zdrojového kódu vlastní aplikace.

3.3.4 Možnosti přenosu streamu

Data, která získáme pomocí FFmpegu, je nutné nějakým způsobem doručit klientům a ještě je zašifrovat. Nabízí se několik možností popsaných níže.

ffserver.exe

Jak již bylo zmíněno výše, jedním z nástrojů, které FFmpeg nabízí, je i streamovací server *ffserver.exe*. Streamovací server slouží k distribuci streamu na více klientů, kteří se k němu připojí. *ffserver.exe* umí z jednoho vstupu vytvořit více různých výstupů. To znamená, že pokud se na něj pošle např. MPEG soubor, tak *ffserver.exe* ho může překonvertovat a klientům nabízet jako třeba *AVI* a *MOV* soubory najednou. *ffserver.c* obsahuje platformě (Linux) závislé komponenty a není ho proto možné přeložit a používat pod systémem Windows. Navíc se i uvádí, že je zastaralý, a nedoporučuje se ho již jako streamovací server používat.

Streamovací servery

Další alternativu pro distribuci streamu představují jiné streamovací servery. Pro naše účely připadají v úvahu ty, které je možné používat zdarma nebo patří do kategorie opensource. Mezi takové patří Darwin Streaming Server (dále Darwin) a C++ RTMP Server (dále crtmp). Oba servery mají vcelku jednoduché ovládání a Darwin disponuje i webovým rozhraním, kde je možné server nastavovat a ovládat.

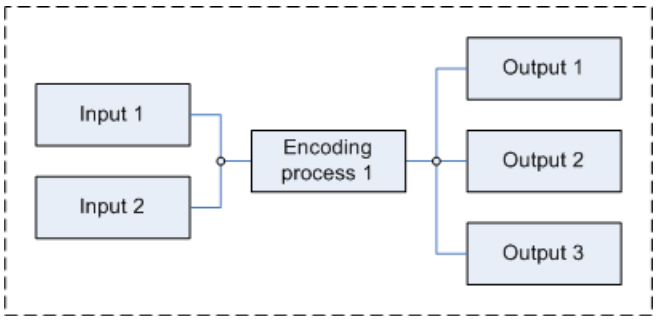
Na Darwinu není problém streamovat soubory, které jsou uložené na disku, ale zdá se být problém streamování live streamu. To naopak není problém u druhého serveru - crtmp. Tento server používá primárně pro přenos streamu protokol RTMP (Real Time Messaging Protocol), ale podporuje i UDP, TCP nebo HTTP. Server crtmp podporuje šifrování streamu v podobě implementovaných protokolů RTMPE (vnitřní bezpečnostní mechanismus) nebo RTMPS (používá SSL/TLS). Aby však tyto protokoly fungovaly, je zapotřebí certifikát vydaný certifikační autoritou. Nestačí použít *self signed* certifikát.

Jako možné řešení je tedy použití streamovacích serverů třetích stran zavrženo.

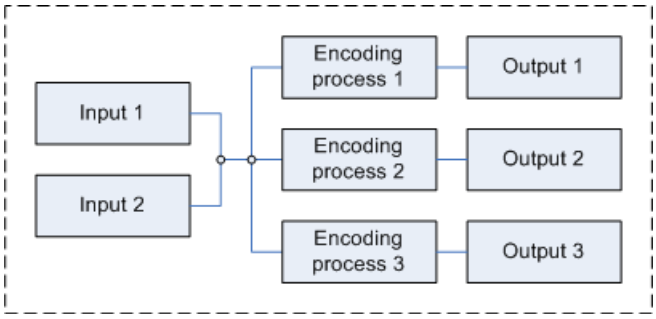
ffmpeg.exe

FFmpeg umí nejen získat data z připojené kamery a mikrofonu, ale dokáže je i posílat (ukládat) na cílovou adresu, která je ve formátu *[protokol]://[ip adresa]:[port]* (závisí na typu protokolu). Není to dlouho, co byla do FFmpegu přidána další funkcionalita v podobě vícenásobného výstupu. Buď je možné jeden překódovaný vstup poslat na více výstupů (viz obr. 3.12) nebo je možné jeden vstup kódovat různými způsoby pro každý výstup zvlášť (viz obr. 3.13), což je výpočetně náročnější [35].

FFmpeg sice umí posílat vstup na více klientů, ale pro naše účely, alespoň co se týče rozesílání streamu klientům, je tento způsob nedostačující. Je to proto, že se bude používat vytvořený nástroj (*ffmpeg.exe*), kde se již při jeho spuštění musí zadat, co se kam bude posílat, a nelze pak za běhu přidávat nové klienty. Jednou možností by bylo pouštět příkaz pro *ffmpeg.exe* pokaždé, když se při(od)pojí nějaký klient. Tím by se ale narušila nezávislost jednotlivých klientů, kteří by na obdrženém streamu mohli zaznamenávat malé výpadky pokaždé, když se někdo jiný odpojí nebo připojí.



Obr. 3.12: Kopírování jednoho vstupu na více výstupů u FFmpegu [35].



Obr. 3.13: Kódování vstupu pro každý výstup zvlášť u FFmpegu [35].

Popsané funkcionality proto využijeme tím způsobem, že sice nebudeme posílat stream pomocí příkazu přímo klientům, ale získaný vstup z kamery (mikrofonu) se bude posílat na lokální porty, kde bude poslouchat server. Z těchto portů bude stream odchytáván a přeposílán na všechny klienty, kteří si o stream zažádali. V případě, že žádný klient neodebírá stream, nebude se ani získávat vstup z kamery (mikrofonu), a tím se ušetří výpočetní čas procesoru. Vzhledem k tomu, že server bude poskytovat 2 streamy (audio+video, audio), budou na serveru existovat 2 lokální porty, na kterých bude server poslouchat a doručený stream přeposílat. Pomocí příkazu pro *ffmpeg.exe* pak budou vytvořeny 2 streamy, kde jeden bude obsahovat jak video tak audio a bude se posílat na lokální port *xxxx*, a druhý bude obsahovat pouze audio a bude se též posílat na (jiný) lokální port *yyyy*. Jak vypadá příkaz pro získání streamu bude popsáno v kap. 4.3.1. Pro přenos streamu byl vybrán protokol UDP, jak již bylo popsáno výše.

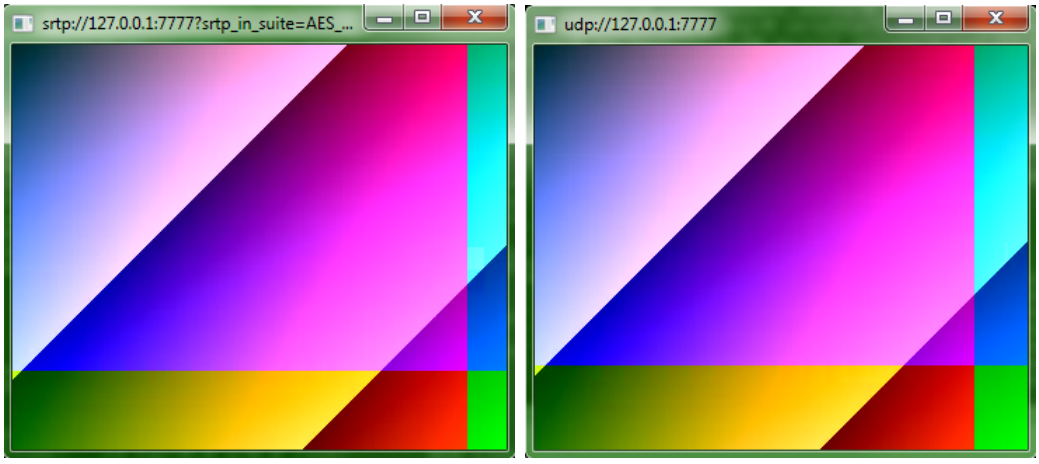
Šifrování

Jako jeden ze základních požadavků na aplikaci je šifrování přenosu kvůli bezpečnosti. Dále se budeme zabývat vytvořením šifrovaného streamu, který budeme získávat výše uvedeným způsobem. FFmpeg podporuje mnoho síťových protokolů, mimo jiné i ty, co umožňují šifrování přenosu. Pomocí protokolu SRTP (Secure Real-time Transport Protocol) můžeme posílat stream např. takto:

```
ffmpeg -re -i "output.mpg"-vcodec libx264 -f flv
srtp://192.168.0.100:7777?srtp_out_suite=AES_CM_128_HMAC_SHA1_80&
srtp_out_params=NmcxMmQ2ZjVnYjEyNmRmMTV2czY1YWR2ZjFhc2Rm
```

(3.1)

Kde *srtp_out_suite* určuje typ výstupního šifrování a *srtp_out_params* je šifrovací blok (kódování base64), kde prvních 16 bytů se použije jako klíč a zbylých 14 jako sůl. Na straně klienta se pak použije stejný šifrovací blok i typ šifrování. Na obrázku 3.14 (a) je ukázka streamu, který byl přenesen pomocí protokolu SRTP a dešifrován stejným klíčem jako je v ukázce 3.1. Na obrázku 3.14 (b) je pak záznam z toho samého streamu, tentokrát ale přijímaného protokolem UDP, který žádné šifrování nepodporuje.



(a) SRTP protokol (b) UDP protokol

Obr. 3.14: Přehrávání šifrovaného streamu pomocí SRTP a UDP protokolů.

Z obrázků je jasně patrné, že protokol SRTP, který je v FFmpegu implementován, žádné šifrování neprovádí. Kdyby ano, zašifrovaný stream nepůjde spustit pomocí protokolu UDP, případně nebude doručený stream čitelný. Parametry byly nastaveny podle dokumentace. Je možné, že se tato chyba vyskytuje pouze na některých buildech a do budoucna bude opravena. Pro šifrování bylo tedy zvoleno jiné řešení, které využívá fakt, že stream, před odesláním klientovi prochází přes server.

Stream, který je pomocí *ffmpeg.exe* posílán na server, je možné ještě před jeho odesláním klientovi zašifrovat. K tomu se použije symetrická šifra AES (viz kap. 3.2.3), kde šifrovací klíč je vygenerován na serveru, a pomocí šifrovaného spojení, které je tou dobou již navázané, doručen klientovi. Klíč je klientovi doručen ještě dřív, než se začne posílat stream, aby klient mohl bez problémů stream dešifrovat.

Pro každý stream bude existovat jeden šifrovací klíč. Šifrovací klíče budou mít omezenou životnost v tom smyslu, že při každém zapnutí příkazu pro odebírání streamu budou klíče vygenerovány znovu. To znamená, že pokud bude stream odebírat jeden klient, který se po nějaké době odhlásí, bude pro šifrování streamu používat jiné klíče, než ten klient, který se přihlásí např. až 5 minut po něm. V případě, že je najednou přihlášeno více klientů, budou všichni používat stejné šifrovací klíče pro šifrování streamu (ale každý klient má jiný klíč pro obousměrnou komunikaci se serverem). Jak se šifrovací klíč pro stream dostane ke klientovi je popsáno v kap. 4.2.2.

3.3.5 Přehrávání streamu

Pro přehrávání streamu na straně klienta byl vybrán nástroj *ffplay.exe* od FFmpegu. Tento nástroj zastupuje multimediální přehrávač a je schopný přehrávat stream jak ze souboru, tak ze sítě. Vzhledem k tomu, že stream, který bude klientovi posílán, bude šifrovaný, není možné ho rovnou směrovat do přehrávače. Na straně klienta se musí vytvořit stejný mechanismus na šifrování streamu jako na serveru. To znamená, že klient bude mít vyhrazený port, na kterém bude přijímat stream. Tento stream projde dešifrovacím procesem, a až pak bude směrován do přehrávače, který ho bude zobrazovat.

3.3.6 Detekce změn

Server musí být schopný analyzovat procházející stream a určit, zda nedošlo ke změně obrazu (pohyb) nebo zvuku (zvýšení hlasitosti). Díky výše popsanému způsobu posílání streamu to je možné. Algoritmy na detekci změny obrazu bývají založené na porovnávání dvou po sobě jdoucích rámců. Neporovnávají je pixel po pixelu, ale vždy po větších částech, a na základě výsledku se pak rozhodují, zda došlo ke změně. FFmpeg má zakomponovánu jak detekci hlasitosti, tak detekci pohybu.

Detekce změny hlasitosti probíhá následovně:

1. Na začátku snímání se do logu zaznamená *silence_start*.
2. Pokud se zvýší hladina zvuku nad mez danou pomocí *dB*, tak se zaznamená *silence_end* a doba trvání ticha.
3. Po uběhnutí určité doby se opět zapíše *silence_start* a začne se měřit čas trvání ticha.

Nás budou zajímat ty úseky logu, kde se vyskytuje *silence_end*, protože to značí, že hladina zvuku se zvýšila nad námi danou tolerovanou mez. Pokud je zapnuté sledování změn scény, pak se do logu zapisují řádky obsahující informaci o tom, jak moc se daný rámec liší od toho předchozího: *scene:0.750000*. Hodnota *0.750000* udává, že aktuální rámec se od toho předcházejícího liší o 75% (před kamerou tedy došlo k většímu pohybu).

Díky možnostem, které FFmpeg nabízí, dokážeme jednoduchým způsobem získat informaci o tom, zda se změnila scéna nebo hladina zvuku. Veškeré tyto informace jsou logovány do konzole, kde *ffmpeg.exe* běží. Naštěstí pro nás, všechny výpisy provádí FFmpeg na standardní chybový výstup, a tak po přesměrování toho výstupu do souboru se lehce dostaneme k obsaženým informacím. Problém zde ale představuje to, že velikost souboru s logem bude při nepřetržitém provozu serveru neustále narůstat. FFmpeg nepodporuje nastavení pevné délky logovacího souboru, kde by se záznamy po dosažení určité velikosti souboru opět zapisovaly od začátku souboru. Z tohoto důvodu musela být zvolena jiná technika pro získání dat, než jen pouhé čtení logovacího souboru.

Pro získání informací z logu je možné použít roury (dále pipe). Logovací soubor tedy vůbec nebude vytvářen a výstup ze spuštěného *ffmpeg.exe* bude pomocí pipe

přesměrován ke zpracování rovnou na server. Podrobnější informace jsou obsažené v implementaci v kap. 4.3.3.

3.4 Správa serveru

Pro správu serveru, streamu a přidávání nových uživatelů bylo, na straně serveru, zvoleno webové rozhraní. Tato varianta je pro běžné uživatele mnohem příjemnější než běžná příkazová řádka, ze které je server spouštěn. Toto webové rozhraní je přístupné přes běžný webový prohlížeč po zadání IP adresy serveru a portu *9443*. Vzhledem k tomu, že se mezi prohlížečem a serverem budou přenášet citlivé informace v podobě uživatelských údajů, je pro komunikaci zvolen protokol HTTPS (viz kap. 3.2.1). Přístupnost webového rozhraní musí být omezena, aby se nemohl kdokoliv registrovat, a pak i sledovat stream. Vstup do administrace je tedy omezen zadáním hesla a uživatelského jména.

3.4.1 Přidání nových uživatelů

Hlavním důvodem vzniku webového rozhraní bylo přidávání nových klientů. Je to kvůli tomu, že protokol SRP (viz kap. 3.2.2) zajišťuje pouze bezpečnou autentizaci, ale už neřeší dopravu citlivých údajů od klienta na server během registrace, která je před samotnou autentizací nutná. Po přihlášení do rozhraní bude tedy možné přidat nového uživatele zadáním jeho uživatelského jména a hesla. Tyto údaje jsou pak uloženy na serveru způsobem, který byl diskutován výše (viz 3.2.2). Uložené údaje pak poslouží k přihlašování klientů na server.

Při zadávání nových údajů bude nutné ohlídat duplicitu uživatelských jmen a také jejich délku. Není možné, aby uživatelským jménem nebo heslem byl prázdný řetězec nebo naopak řetězec přesahující 300 znaků. Stejně tak je nutné omezit znaky, které je možné pro uživatelské údaje použít. Ne všechny znaky se přenesou tak, jak je uživatel zadá, u některých dochází k převedení do hexadecimálního kódu (např. znak *!* se přenese jako *%21*). Tomu je nutné zabránit, uživatel by se pak nikdy nemohl pod registrovanými údaji přihlásit, protože by zadával úplně jiné údaje, než jaké jsou ve skutečnosti uloženy.

3.4.2 Nastavení serveru

Webové rozhraní se dá využít, kromě registrace nových uživatelů, i k základní správě serveru, kterou může provádět uživatel. Mezi tyto úkony patří změna portů, na kterých běží jednotlivé části serveru, změna hlavního hesla do rozhraní a omezení přístupu do rozhraní. U portů je nutné ohlídat, aby se jejich hodnota nacházela mezi čísly *1025 a 65535* a aby nebyly duplicitní.

Změna hesla do administrátorské části serveru má stejná pravidla jako většina takových změn stejného typu. Pro změnu hesla je nutné zadat i to staré, aby bylo ověřeno, že ho uživatel zná. Délku hesla je opět nutné omezit, aby uživatel nezadával příliš krátká/dlouhá hesla. Omezení se týká i skupiny použitých znaků, která je stejná jako výše (viz kap. 3.4.1). Nové heslo se pak uloží jen v případě, že to staré se bude shodovat s aktuálně uloženým. Přístupové údaje do administrátorské části webového rozhraní jsou na server uloženy jako hashe.

Čím méně bude webové rozhraní přístupné, tím menší je šance, že se do něj útočník dostane. Přístup do rozhraní je možné omezit tak, že budou přijímáni pouze weboví klienti ze stejné podsítě, ve které je sám server. Toho lze dosáhnout pomocí masky podsítě.

Maska sítě

Síťová maska se používá pro určení adresy sítě (bitů, které jí přísluší), která je součástí IP adresy [9]. Jedná se o čtyřbytové číslo, které má (ve dvojkové soustavě) zleva v místech síťové adresy samé *1*, jinak *0* [9]. IP adresa klienta do podsítě patří, pokud logický součin masky podsítě a IP adresy klienta je roven IP adrese podsítě. V ukázce 3.2 je uveden příklad, kde se podle IP adresy serveru nejdříve určí IP adresa podsítě a následně se podle ní posoudí, zda IP adresa klienta je vyhovující.

```
maska sítě: 255.255.254.0₁₀ = 1111 1111.1111 1111.1111 1110.0000 0000₂
IP serveru: 147.228.187.72₁₀ = 1001 0011.1110 0100.1011 1011.0100 1000₂
IP klienta: 147.228.67.109₁₀ = 1001 0011.1110 0100.0100 0011.0110 1101₂

   Provedeme logický součin masky a IP adresy serveru a získáme IP ad-
```

resu podsítě:

$$\begin{array}{ll} & 11111111.11111111.11111110.00000000 \\ \wedge & \underline{10010011.11100100.10111011.01001000} \\ & 10010011.11100100.10111010.00000000 = 147.228.186.0_{10} \end{array}$$

Druhý logický součin provedeme s maskou a IP adresou klienta:

$$\begin{array}{ll} & 11111111.11111111.11111110.00000000 \\ \wedge & \underline{10010011.11100100.01000011.01101101} \\ & 10010011.11100100.01000010.00000000 = 147.228.66.0_{10} \end{array}$$

Nyní porovnáme prvních 23 bitů (počet výskytů 1 v masce zleva) (odděleny |) obou výsledků, které nás zajímají:

$$\begin{array}{l} 10010011.11100100.1011101|0.00000000 \\ 10010011.11100100.0100001|0.00000000 \end{array}$$

Jak je vidět, IP adresy podsítí se liší, a tím pádem IP adresa klienta nepatří do dané podsítě.

(3.2)

Pokud bude IP adresa klienta po odmaskování patřit do stejné sítě jako IP adresa serveru pak se klient bude moci na server přihlásit. Tuto možnost je lepší nechat volitelnou, aby si uživatel mohl sám vybrat, zda se do webového rozhraní dá dostat pouze ze stejné podsítě či odkudkoliv.

Změna nastavených portů se projeví až po restartování serveru, není možné změnit porty za běhu. Změnu hesla administrace a nastavení podsítí je možné aplikovat okamžitě. V případě, že uživatel bude přistupovat např. z IP adresy jako ve výše zmíněném příkladu a zakáže přístup z jiné podsítě, pak se mu po uložení změn stane, že se již do webového rozhraní nepřihlásí.

3.4.3 Nastavení streamu

Další funkcionalitu, kterou může webové rozhraní nabízet, je nastavení přenášeného streamu a nastavení parametrů pro detekci změn. Toto nastavení je pro všechny uživatele, kteří budou přijímat stream, stejné. V případě nastavení přenášeného streamu se uživateli může zpřístupnit nastavení kvality jako např. rozlišení, počet zvukových kanálů (mono, stereo), kvalita videa nebo audio frekvence. Vzhledem k tomu, že na těchto hodnotách závisí i úspěšné spuštění příkazu pomocí *ffmpeg.exe*, nebude si je moci uživatel zadávat libovolně. Místo toho bude mít na výběr z několika přednastavených hodnot, např. v podobě výběrového pole (*select box*). Dalším důsledkem, který z toho plyne je, že změna nastavených hodnot se změní až po tom, co dojde k znovu spuštění příkazu pomocí *ffmpeg.exe* (tzn. po vypnutí všech odebíraných streamů).

Posílání zpráv o změnách hlasitosti (změny obrazu) klientům se řídí na základě zadaných mezí. Pokud jsou tyto meze překročeny, je poslána zpráva o změně. Ve webovém rozhraní je tedy možné zpřístupnit nastavení těchto mezí. Jedna mez se týká hlasitosti zvuku a je udávána v *dB*. Krajní hodnoty budou nastaveny na $0dB$ (práh slyšitelnosti) a $120dB$ (start tryskového letadla [18]). Výchozí hodnota pak bude nastavena na $40dB$, což odpovídá tiché knihovně [18] (Záleží i na nastavení citlivosti mikrofonu).

Druhou nastavitelnou mezí je míra změny scény vlivem pohybu. Tato hodnota je udávána v procentech a její hranice tak jsou od 0% (detekce pohybu při změně 0% scény) do 100% (detekce pohybu při změně 100% scény). Výchozí hodnota, která stačí pro detekování změny, je pak nastavena na 10%.

Změna meze pro hladinu zvuku je vnitřně využívána přímo *ffmpeg.exe*. Pro uživatele to znamená, že změny meze se projeví až po tom, co všichni přihlášení klienti vypnou běžící streamy. Při dalším spuštění streamu dojde k novému zapnutí *ffmpeg.exe* s již nově nastavenou hodnotou. Toto však neplatí pro mez detekce pohybu, jelikož log z *ffmpeg.exe* obsahuje pouze informaci o tom, jak moc se scéna změnila (viz kap. 3.3.6). Tím pádem se o posílání zpráv o změně scény musí rozhodnout až vnitřně na serveru a ten má k dispozici vždy aktuální hodnotu, s kterou získaná data porovnává. Uložení nové meze pro detekci pohybu se tedy projeví hned i bez vypínání streamů.

4 Programátorská příručka

V této kapitole se budeme věnovat implementaci jednotlivých částí serveru a klienta.

4.1 Architektura aplikace

Jak pro server, tak pro klienta, bylo zvoleno objektové programování pomocí tříd a programovacího jazyka *C++*. Třídy, které provádějí určitý typ činnosti, jsou sdružovány do souborů. Aplikace musí být platformně nezávislá[1]. Za tímto účelem byl v obou částech vytvořen soubor *platform.h*, který obsahuje mapování funkcí, které jsou pro oba operační systémy rozdílné. Mimo to se v kódu objevují i logické celky ohraničené pomocí direktiv (*#ifdef, #endif*), které jsou pro oba systémy odlišné. Další soubor, který mají obě části aplikace, je *config.h*, který obsahuje veškeré globální proměnné.

Jak server, tak klient, obsluhují více běžících vláken najednou. Za účelem usnadnění a sjednocení práce s vlákny pod oběma systémy bylo použito rozhraní pro psaní vícevláknových programů pro systémy Windows - POSIX Threads for Win32 (pthreads) [22]. Díky tomuto rozhraní je možné pracovat s vlákny stejně pod oběma systémy, bez nutnosti použití direktiv *#ifdef, #endif atd.*

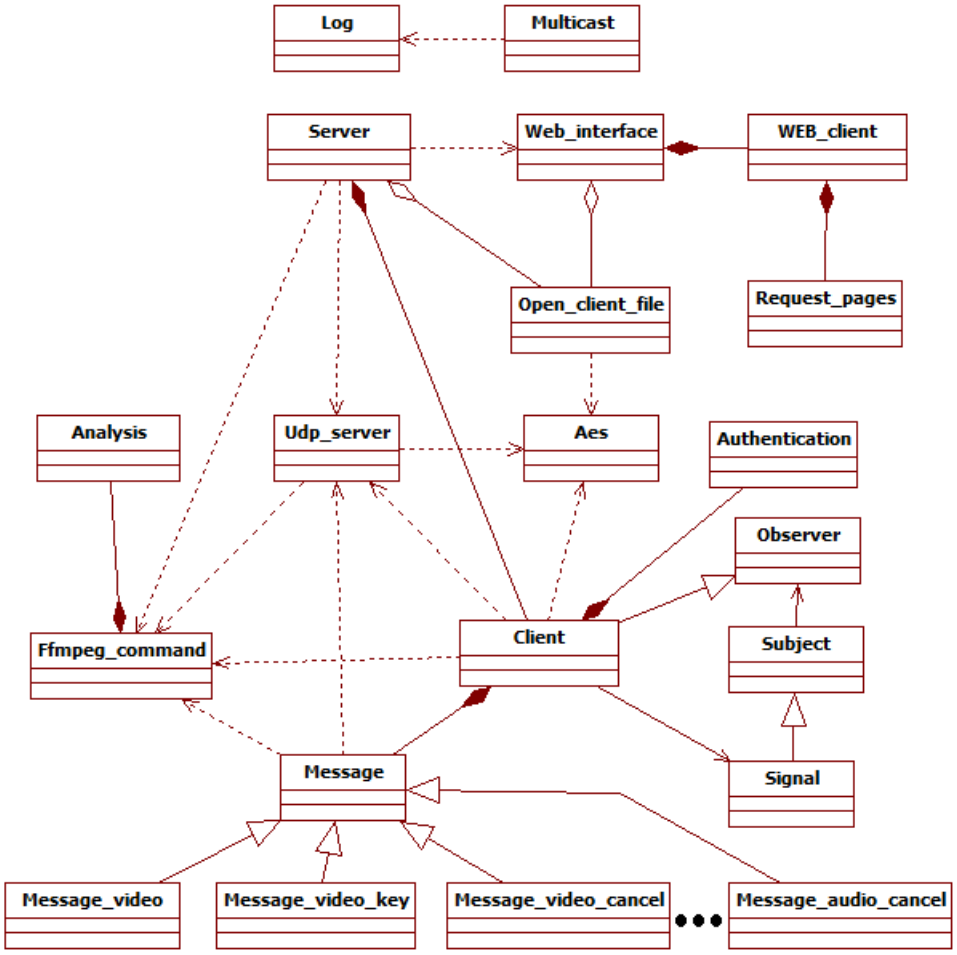
4.1.1 Server

Funkcionalita serveru byla rozdělena do několika logických celků (*client, ffmpeg, log, multicast, udp, web_interface, hlavní*). Ten hlavní obsahuje vytvoření serveru, sledování vstupu z klávesnice, konfigurační soubor a soubor s funkcí *main*. Celek *client* obsahuje veškerou logiku, co se týče přihlášení, komunikace a obsluhy klienta. Jsou zde i níže popsané třídy pro práci se zprávami. Dále je zde uložen i zašifrovaný soubor s registrovanými uživateli.

[1] Aplikace musí být přeložena pod systémy Windows i GNU/Linux

ffmpeg obsahuje veškerou logiku, co se týče práce se streamem a detekce změn. *log* obsahuje logovací třídu, která se používá ve všech ostatních třídách a umožňuje jednotný zápis logu do souboru. *web_interface* obsahuje třídy, které pracují s webovým rozhraním, certifikáty a textové soubory s nastavením (serveru, FFmpegu, změn).

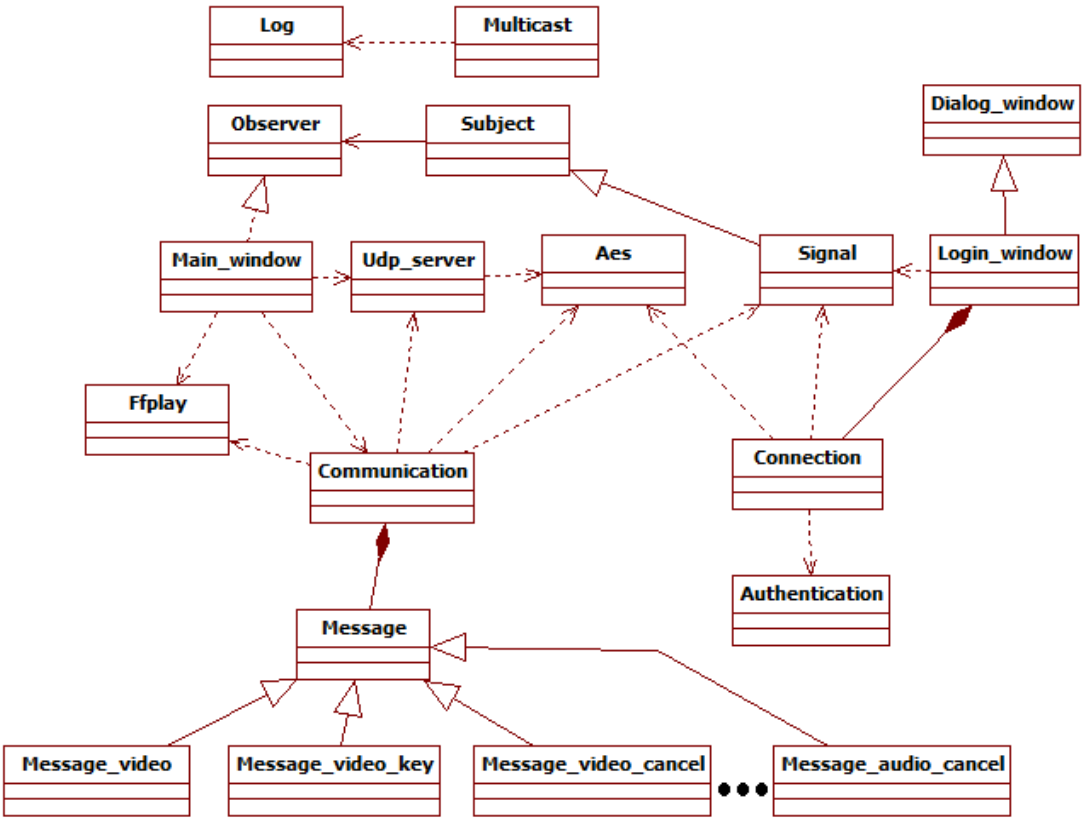
Detailnější provázanost jednotlivých tříd je možné vidět v UML diagramu tříd na obr. 4.1. Tento diagram je zjednodušený, tedy neobsahuje úplné popisy tříd v podobě atributů a metod. Dále je zde, kvůli přehlednosti, vynechána vazba všech tříd na třídu *Log*, kterou využívají k logování, a výpis tříd, které se používají na zpracování zpráv, je zkrácen pomocí značky $\cdots$.



Obr. 4.1: UML diagram tříd pro server.

4.1.2 Klient

Třídy v programu klienta byly rozděleny stejným způsobem jako na serveru, podle funkčních celků. Ty jsou téměř shodné, navíc se zde objevuje *gui*, které obsahuje třídy pro práci s grafickým rozhraním, a místo *ffmpeg* tu je *ffplay*, kde je soustředěno přehrávání streamu. UML diagram je znázorněn na obr. 4.2, kde došlo ke stejnému zjednodušení, jako u diagramu tříd serveru.



Obr. 4.2: UML diagram tříd pro klienta.

4.2 Zabezpečení

Implementace registrace nových klientů bude popsána v kap. 4.4.1.

4.2.1 Přihlášení

Pro přihlášení klientů na server byl zvolen protokol SRP, který je popsán výše (viz kap. 3.2.2), a jehož implementace byla převzata z [29] a následně nepatrně upravena podle požadavků aplikace. Tento protokol musí být implementován na obou komunikujících stranách, jinak by přihlášení nebylo možné. Dále si přiblížíme proces přihlášení na obou stranách.

Klient

Po vyplnění údajů do přihlašovacího okna dojde nejdříve k jejich zkontrolovaní, zda nejsou nějaká pole prázdná nebo zadané hodnoty nejsou nesmyslné. Následně se provede pokus o navázání spojení se serverem podle zadané IP a portu. Pokud se spojení naváže, spustí se autentizační proces se zadaným uživatelským jménem a heslem. Výsledkem úspěšné autentizace je relační klíč, který bude dále použit (viz kap. 4.2.2), a zapnutí okna s přehrávačem. V případě, že se spojení nenaváže nebo jsou zadány špatné údaje, přihlašovací okno se otevře znovu a vypíše, jaká nastala chyba.

Server

U nově příchozího klienta se nejdříve rozhodne, zda se jedná o normálního klienta nebo o webového klienta. V případě normálního klienta server vytvoří v novém vláknu instanci třídy *Client*, ve kterém bude klient obsluhován. V tomto vláknu je nejdříve spuštěn autentizační proces. Server si po spuštění načte všechny registrované uživatele s jejich údaji (hash, sůl, jméno) do paměti. V těchto údajích pak hledá uživatelské jméno, které obdrží od klienta, a sůl a hash pak používá v průběhu autentizačního procesu. Pokud proces skončí neúspěchem, vlákno je ukončeno. V případě úspěchu je spuštěn cyklus, ve kterém se vyčkává na zprávy od klienta, a dále je vytvořeno vlákno, které bude klientovi zprávy posílat. Stejně jako u klienta, i zde vznikne relační klíč.

4.2.2 Získání šifrovacích klíčů

Výsledkem úspěšného autentizačního procesu, u klienta i serveru, jsou relační klíče, s nimiž je možné šifrování komunikace. My je však nepoužijeme k šifrování, ale pouze k vygenerování nových šifrovacích klíčů pomocí knihovny OpenSSL (viz [21]). Tato knihovna obsahuje funkci, která generuje šifrovací klíče ze zadaného základu. Klíče, generované ze stejného základu, jsou pokaždé stejné, a to je i důvod, proč nás nemusí trápit distribuce klíče u symetrické šifry AES (viz kap. 3.2.3). Na obou komunikujících stranách máme díky úspěšnému autentizačnímu procesu stejné relační klíče, které mohou být použity jako základy pro vygenerování nových klíčů (256 bitů).

Knihovna poskytuje pro AES dvě šifrovací metody, *CBC* a *ECB*. Kvůli výše diskutovaným důvodům (viz kap. 3.2.3) byla použita metoda *CBC* pro veškeré šifrování přenosu. Je důležité zdůraznit, že každý klient má jiné šifrovací klíče pro komunikaci se serverem, a stejně tak jsou klíče jiné při každém novém přihlášení. Kromě těchto klíčů existují ještě další dva klíče, které ale mají všichni klienti společné, sdílí je. Jedná se o klíče určené k šifrování videa a audia. Tyto klíče vznikají vždy až v okamžiku, když je zapnut odběr streamu. Pokud si klient vyžádá stream, jsou mu tyto klíče poslány pomocí šifrovaného spojení, které je již navázané.

Životnost těchto klíčů se váže na odběr streamu. Klíče se vytváří nové vždy, když je spuštěn příkaz na odběr streamu. Prakticky to znamená, že pokud se přihlásí klient k odběru streamu, jsou vygenerovány nové klíče. Každý další klient, který se přihlásí, dostane též tyto klíče. Pokud se všichni odhlásí, a následně se jeden znovu přihlásí k odběru streamu, pak už bude mít klíče nové.

4.2.3 Multicast

Pro vytvoření mDNS služeb byla, s malými úpravami, použita implementace podle [32], kde bylo nutné dopsat posílání odpovědí zpět klientovi. Tento program funguje tak, že se nejdříve do záznamu uloží pár *IP adresa-název stroje*, a pak se už jen přijímají požadavky od klientů. Pokud se název stroje, obsažený ve zprávě od klienta, shoduje s tím, co je uložen v záznamech, pošle se klientovi odpověď, která obsahuje IP požadovaného stroje, jinak se neposílá nic.

Implementace je napsána v programovacím jazyce C, je přeložitelná pod systémy Windows i GNU/Linux a byla použita jako přeložený program, který je spouštěn jako externí proces z programu server. V tomto programu je naprogramována nekonečná smyčka, která čte vstupy z klávesnice, a pokud se zadá písmeno *q*, pak je program ukončen. V případě, že by tento způsob z nějakého důvodu nefungoval, pak je ještě implementován příkaz *kill*, který tento externí proces vyhledá mezi spuštěnými procesy a ukončí ho.

mDNS funguje pouze, pokud jsou server i klient ve stejné podsíti. Jinak je nutné použít IP adresu serveru pro připojení z klienta na server.

4.2.4 Komunikace

Jak už bylo zmíněno výše (viz kap. 3.2.3), pro komunikaci mezi serverem a klientem byl použit protokol TCP a pro posílání streamu pak protokol UDP. Nyní se budeme věnovat obousměrné komunikace *klient-server*.

Požadavky, které má klient na server (ale i opačně, server může chtít informovat klienta o změně) jsou řešeny pomocí zasílání zpráv ve formátu *###type#state#data*, kde *type* je typ zprávy (viz tabulka 4.1), *state* indikuje úspěšnost provedení požadavku (0 - selhání, 1 - úspěch) a *data* obsahuje dodatečná data, která je třeba případně poslat (používá se pro přenos šifrovacího klíče streamu). Pro každý typ existuje samostatná třída, která daný úkol provádí. Všechny tyto třídy dědí od jedné společné a překrývají funkci pro zpracování zprávy.

Všechny zprávy jsou před svým odesláním šifrovány pomocí AES, kde délka bloku je 16 bytů. Možné délky zpráv po zašifrování tedy odpovídají násobkům 16. Od klienta na server se nikdy neposílá zpráva delší než 16 bytů. Typy zpráv a jejich význam je popsán v tabulce 4.1. Pokud klient pošle zprávu na server, tak ten zachová typ zprávy a mění pouze pole *state* a případně *data*.

#	Název (typ)	Popis funkce
0	VIDEO	Klient pošle v okamžiku, kdy si chce nechat začít posílat video stream. V části pro data posílá svůj UDP port, kde bude poslouchat. Server naopak odpovídá se svým portem, odkud bude data posílat. Nyní je možné posílat stream klientovi, který je uložen v seznamu odběratelů.
1	AUDIO	To samé jako VIDEO, ale pro audio.
2	FRAME_-CHANGE_-SUBSCRIBE	Klient chce dostávat zprávy o změnách obrazu. Vždy, když se změní obraz, bude mu poslána zpráva.
3	SOUND_-CHANGE_-SUBSCRIBE	Stejné jako FRAME_CHANGE_SUBSCRIBE, ale pro změnu hlasitosti.
4	CANCEL_-VIDEO	Tato zpráva je poslána, pokud vyprší čas určený k dočasnému přehrání videa. Přenos streamu na daného klienta je ukončen.
5	CANCEL_-AUDIO	Stejné jako CANCEL_VIDEO, ale pro audio.
6	KEY_VIDEO	Klient posílá žádost o zaslání šifrovacího klíče pro video stream. Server odpovídá stejnou zprávou s vloženým klíčem v části pro data. Klient je zařazen do seznamu, kterému je posílán stream, ale zatím se mu nic neposílá (není znám port, kam se má stream posílat).
7	KEY_AUDIO	Analogie ke KEY_VIDEO.
8	FRAME_-CHANGE	Tuto zprávu posílá server klientovi v případě, že nastala změna obrazu. Klient na ní neodpovídá, pouze provede příslušné kroky k zobrazení zprávy.
9	SOUND_-CHANGE	Stejné jako FRAME_CHANGE.
10	QUIT_-SERVER_-MESSAGE	V případě, že bude server ukončen, pošle ještě předtím zprávu, aby se klienti mohli taktéž řádně ukončit.
11	STOP_-CLIENT	Zpráva přijde od klienta, který zastavil přehrávání streamu pomocí tlačítka *Stop*. Na serveru se klient odstraní ze všech seznamů, které souvisí s posíláním streamu a zpráv o změnách.

Tab. 4.1: Typy a význam komunikačních zpráv.

4.3 Stream

4.3.1 FFmpeg

Nástroj *ffmpeg.exe* byl blíže přiblížen v kap. 3.3.4. Na možnosti začlenění do aplikace se podíváme nyní. *ffmpeg.exe* je spouštěn jako externí program v novém vláknu, který je však možné z programu serveru ovládat. Toho je u obou platforem docíleno pomocí pipe. Díky tomu je možné *ffmpeg.exe* ukončit legálním způsobem, a to tak, že je mu zasláno písmeno *q*.

Předtím, než je možné příkaz spustit, je nutné znát jeho parametry a hlavně vstupní zařízení, ze kterých se bude odebírat audio/video. V případě systému Windows se počítá s použitím vstupních zařízení pomocí formátů DirectShow (multimediální framework). Tato zařízení je ale nejdříve nutné najít. To se provede spuštěním příkazu 4.1, kde jeho zkrácený výstup je uveden v ukázce 4.2. Tento výstup je směrován do souboru, odkud se pak zařízení získají. Tato zařízení se pak použijí jako vstupní zdroje streamu. Pro systém GNU/Linux se používá vstupní formát *video4linux2* a zařízení */dev/video0* pro získání videa a formát *alsa* a zařízení *default* pro zisk audia. Zde není třeba zařízení vyhledávat.

```
ffmpeg.exe -list_devices true -f dshow -i dummy
```

(4.1)

```
[dshow  02509800]  DirectShow video devices
[dshow  02509800]    "1.3M HD WebCam"
[dshow  02509800]  DirectShow audio devices
[dshow  02509800]    "Mikrofon (Conexant High Definit"
dummym: Immediate exit requested
```

(4.2)

Někdy se může stát, že zařízení obsahuje české znaky a to je problém. FFmpeg spuštěný pod Windows je může špatně dekódovat a tím pádem zařízení nenajde a

vůbec se nespustí. Kvůli tomu je nutné příkaz pro spuštění nejdříve překódovat z UTF-8 na Windows ANSI a až pak ho je možné spustit.

Nyní je možné pomocí nástroje *ffmpeg.exe*, spustit příkaz, který zapne získávání videa a audia z dostupných zařízení. Dále pak získaný stream kóduje do formátu *MPEG-TS* a kopíruje ho na dva výstupy, které jsou posílány na lokální porty serveru, kde jeden výstupní stream obsahuje jak video, tak audio stopu, a druhý pouze audio stopu. Kromě toho je spuštěna i detekce překročení limitu hladiny zvuku a detekce pohybu. Výstupní log je přesměrován do pipy, aby ho bylo možné analyzovat (viz kap. 4.3.3). Celý příkaz (pro Windows) je znázorněn v ukázce 4.3 (spolu s číslováním řádků) a v tabulce 4.2 jsou pak vysvětleny jeho jednotlivé parametry.

```
1: ffmpeg.exe -loglevel debug
2: -f dshow
3: -i video="1.3M HD WebCam":audio="Mikrofon (Conexant High Definit"
4: -vcodec libx264 -pix_fmt yuv420p
5: -acodec libmp3lame
6: -b:v 282000 -ar 16000 -ac 1 -r 30 -s 800x600 -q:v 1
7: -f mpegts "udp://127.0.0.1:9002"
8: -af silencedetect=noise=-40dB -f null -
9: -vf "select='gt(scene 0.1)'"-f null -
10: -vn -f mpegts "udp://127.0.0.1:9004"
11: 2>\\.\\pipe\\ffmpegOutputPipe
```

(4.3)

Na řádcích 7 - 10 jsou vidět paralelní výstupy. Řádek 7 představuje audio+video stream, který je ve formátu *MPEG-TS* posílán na lokální port serveru. Řádek 10 ukazuje to samé, jen je posílán pouze audio stream a na jiný port. Na řádcích 8 a 9 jsou nastaveny filtry pro detekci hlasitosti a pohybu. Za normálních okolností poskytují výstup (další stream nebo např. obrázky při každé detekci pohybu), ten však my k ničemu nepotřebujeme, a proto je zde místo výstupního formátu pouze *null* -. Řádky 2 a 3 závisí na použitém operačním systému. Poslední řádek je přesměrování chybového výstupu do pojmenované pipy. Díky tomu je možné analyzovat veškerý log, který je generován.

Parametr	Význam
loglevel	Určení úrovně logování. *debug* je nejpodrobnější výpis.
f	Vynucení určitého v(ý)stupního formátu. *dshow* značí použití zařízení DirectShow. *mpegts* je formát výstupního streamu.
i	Formát vstupu. Zde je uveden buď vstupní soubor, síťová adresa nebo zařízení, jako výše v ukázce 4.3.
vcodec	Kodek pro video. *libx264* je knihovna a značí použití komprese *H.264*.
pix_fmt	Formát rámců. Konkrétně použitý je *yuv420p* (viz obr. 3.10).
acodec	Kodek pro audio. *libmp3lame* je knihovna a značí použití komprese *MP3*.
b:v	Video bit rate. Jedná se o počet přepravených bitů za jednotku času.
ar	Vzorkovací frekvence audia. Nejpoužívanější hodnoty jsou 44 100 *Hz* a 22 050 *Hz*.
ac	Počet audio kanálů (mono = 1, stereo = 2).
r	Počet rámců za sekundu.
s	Rozlišení videa.
q:v	Určuje míru komprese - kvalitu videa. Rozpětí má od 1 do 31, kde 1 je nejkvalitnější.
af	Audio filtr. *silencedetect=noise=-40dB* udává sledování hladiny zvuku s tolerancí 40 *dB*.
vf	Video filtr. *select='gt(scene 0.1)'* porovnává dvě po sobě jdoucí scény a zaznamená, o kolik procent se liší.

Tab. 4.2: Popis parametrů FFmpegu [12].

Spuštění *ffmpeg.exe* se provádí stejně jako u spuštění mDNS programu, pomocí pipe. Příkaz pro získávání a analýzu streamu neběží celou dobu, co je spuštěn server. Kvůli šetření operační paměti je tento příkaz spuštěn jen tehdy, když si nějaký klient vyžádá stream (po stisku tlačítka *Play* na přehrávači). Po odpojení je vykonávání příkazu přerušeno a s nově příchozím klientem se znovu spouští.

4.3.2 Přenos streamu

ffmpeg.exe posílá na server 2 streamy pomocí protokolu UDP na dva různé porty (oba streamy jsou posílány pokaždé, bez ohledu na to, zda si klient vyžádal pouze audio nebo audio+video). Na těchto portech server poslouchá. Respektive na nich

poslouchají instance třídy *Udp_server*, které běží v samostatných vláknech. Tyto instance mají za úkol přijímat stream, šifrovat ho a posílat na všechny klienty, kteří jsou uloženi v příslušných seznamech (zažádali si o stream). V těchto seznamech se uchovávají IP adresy a porty klientů, kde poslouchá jejich UDP server. Pokud jsou seznamy prázdné, pak není spuštěn ani *ffmpeg.exe* ani vlákna UDP serverů.

Na straně klienta je vždy pouze jedna instance UDP serveru, jelikož klient může přijímat vždy maximálně jeden stream. Zde má UDP server opačnou roli než na serveru. Přijímaný stream dešifruje a posílá ho na lokální port klienta, kde poslouchá přehrávač *ffplay.exe*, který přijatý stream zobrazuje.

4.3.3 Přesměrování logu

Aby bylo možné dále pracovat s detekovanými změnami hlasitosti a obrazu, které jsou logovány, je nutné získat k tomuto logu přístup. Toho bylo dosaženo pomocí pipe, kde pro systém Windows byly použity pojmenované pipy (viz 12. řádek v ukázce 4.3) a pro systém GNU/Linux obyčejné. Díky pipám je možné log přesměrovat z výstupu na konzoli ke zpracování na server. Na serveru je pak pomocí instance třídy *Analysis*, která běží v samostatném vláknu, nepřetržitě čten a analyzován výstup pipe (veškerý log, který *ffmpeg.exe* produkuje).

4.3.4 Detekce změn

Veškerá detekce změn probíhá na základě analýzy přesměrovaného logu. Pro detekci změny hlasitosti jsou klíčové ty řádky logu, kde se vyskytuje spojení *silence_end*, a pro detekci pohybu pak řádky se slovem *scene* (viz kap. 3.3.6). V případě, že je nalezen řádek se změnou, jsou pomocí signálů uvědoměni všichni klienti (vlákna na serveru spravující připojené klienty). Pokud má dané vlákno, obsluhující klienta, zaznamenáno, že klient (druhá část aplikace) chce odebírat zprávy o změnách, pak mu pošle zprávu (viz kap. 4.2.4), jinak oznámení ignoruje. Pro zasílání signálů informujících o detekci zvuku byl použit návrhový vzor *Observer*.

Detekované změny pomocí *ffmpeg.exe* jsou bez dalších zkoumání použity jen v případě hlasitosti. Co se týče detekce pohybu, z logu jsou vyextrahovány informace

o procentuální změně scény. Až na serveru se pak určuje, zda je tato změna menší (větší) než zadaná mez. Díky tomu je také možné měnit tuto mez přes webové rozhraní a změna se hned projeví. To neplatí o změně meze pro hlasitost. Tu detekuje přímo *ffmpeg.exe*, a proto je po její změně ve webovém rozhraní nutné nástroj *ffmpeg.exe* vypnout a znovu zapnout.

4.3.5 Přehrávání streamu

Pro přehrávání streamu na straně klienta byl zvolen nástroj *ffplay.exe*. Tento program se pod Windows spouští v konzoli pomocí *ShellExecute*, kde se nastavením příslušného parametru zamezí zobrazení konzolového okna. Pod systémy GNU/Linux se provede vytvoření nového procesu (pomocí *fork*), a v tom je spuštěn přehrávač (*exec*). Přehrávač přijímá již dešifrovaný stream z lokálního portu klienta, kam je tento dešifrovaný stream posílán.

Přehrávání streamu nezačne hned po zapnutí přehrávače. Je zde určitá prodleva (cca 10 sekund), protože přehrávači chvíli trvá, než v příchozím streamu nalezne potřebné informace pro přehrávání. V případě velkého vytížení procesoru se ztrácí i více UDP paketů, a pak je nalezení informací o to obtížnější. Pokud do určité doby informaci nenajde, přehrávač se ukončí. Tato funkcionalita je zabudována přímo v přehrávači. Stejně tak se může stát, že je vypnut kvůli nějaké chybě, která při analýze streamu nastala.

Přehrávač otevírá okno se streamem vždy, i v případě, že přehrávaný stream je pouze audio. V tomto okně se pak zobrazuje průběh zvukových vln. Existuje zde parametr *nodisp*, který zamezí otevření okna, a doručený audio stream je pak možné pouze poslouchat.

ffplay.exe obsahuje několik vnitřních příkazů pro ovládání streamu, které jsou nezávislé na naší aplikaci. Z těch, které fungují, to jsou klávesy:

- *q, ESC* - vypnutí přehrávače
- *f* - přepnutí na celou obrazovku
- *p, mezerník* - pauza

- *s* - procházení videa rámec po rámci, předtím se musí zmáčknout pauza
- *šipka doprava* - posunutí na aktuální pozici v live streamu
- *w* - přepnutí na zobrazení zvukových vln

Pokud se *ffplay.exe* vypne použitím výše uvedených kláves, přehrávač se to nedozví, a je nutné ho zastavit ještě příslušným tlačítkem *Stop*. Pokud se přehrávač vypíná přes tlačítko *Stop*, pak je proces *ffplay.exe* ukončen pomocí příkazu *kill*.

4.3.6 Ukládání streamu

Jako rozšíření aplikace bylo naprogramováno ukládání přehrávaného streamu do souboru na straně klienta. Stream bylo možné ukládat tak, jak přicházel na UDP server. Tato možnost se ale projevila jako nevyhovující, jelikož soubor pak nebylo možné přehrát. Byla tedy zvolena jiná možnost a to použití nástroje *ffmpeg.exe* . V případě, že uživatel zaškrtl ukládání streamu do souboru, pak je dešifrovaný stream z UDP serveru posílán jednak do *ffplay.exe* a jednak i na port, kde poslouchá *ffmpeg_save_file.exe*, což je přejmenovaný *ffmpeg.exe*. (viz obr. 3.1)

ffmpeg.exe převede stream z *MPEG-TS* na *MPEG* a uloží ho do souboru s jedinečným jménem ve tvaru *streamFile_YYYY-mm-dd_HH-MM-SS.mpg* (např. streamFile_2014-04-24_10-52-59.mpg). *ffmpeg.exe* je vypnut po stisku tlačítka *Stop*. Stream je možné ukládat jen v případě, že je zapnuté souvislé přehrávání, ne při změnách.

4.4 Správa serveru

Správa serveru probíhá přes webové rozhraní, které je přístupné přes IP adresu serveru a port *9443*. Na tomto portu poslouchá instance třídy *Web_interface*, která má za úkol přijímat nové klienty pomocí protokolu *HTTPS*, k jehož implementaci používá knihovnu OpenSSL (viz [21]). Tato instance umí též filtrovat příchozí klienty podle toho, zda patří do stejné podsítě, jako server. Toto se děje na základě odmaskování klientovi IP adresy a porovnání výsledku s IP podsítě serveru (viz příklad 3.2) [6].

Každý příchozí klient dostává přiděleno vlákno, ve kterém se obslouží jeho jeden požadavek a vlákno poté zaniká. Vzhledem k tomu, že klienti se do webového rozhraní musí přihlásit, je důležité si nějakým způsobem zaznamenávat, kteří již mají schválený přístup a kteří nikoliv. Toho bylo dosaženo pomocí coockie. Při prvním požadavku klienta (webového prohlížeče) na stránku rozhraní se vygeneruje jedinečné identifikační číslo, které je vloženo do *HTTPS* hlavičky, která se spolu s *HTML* kódem stránky posílá zpět klientovi. Klient si toto číslo uloží a při každém dalším požadavku se prokazuje tímto číslem.

Pokud se klient úspěšně přihlásí do rozhraní (pro přístup je jen jedno společné heslo), pak je jeho číslo zařazeno do seznamu přihlášených (ověřených). Při každém dalším požadavku na stránku se musí kontrolovat, zda je uživatel již přihlášen a má tedy právo na zobrazení dané stránky. Po odhlášení se identifikátor smaže ze seznamu přihlášených.

HTML kód, který se přenáší na klienta, je předpřipraven a uložen v proměnných v redukované podobě. Tím je dosaženo zmenšení objemu přenášených dat na úkor přehlednosti kódu. *HTML* kód totiž díky kompresi neobsahuje žádné přebytečné mezery nebo odřádkování.

4.4.1 Registrace

Pokud se klient úspěšně přihlásí do webového rozhraní, má možnost přidat nového uživatele. Za tímto účelem existuje na stránce *https://<IP serveru>:9443/user.html* formulář. Tento formulář obsahuje dvě textová pole, jedno pro uživatelské jméno a druhé pro heslo. Obě dvě pole mají omezený počet znaků na 6-50 včetně a omezenou množinu použitelných symbolů (a-z A-Z 0-9 *_.-).

Vyplněné údaje jsou poslány na server, kde se kontroluje, zda už takové uživatelské jméno existuje, pokud ano, uživatel se nepřidá. Pokud se jedná o nového uživatele, pak je uložen jednak do seznamu registrovaných uživatelů, který je přístupný po celou dobu běhu serveru, a je tedy možné nově registrované údaje hned použít k přihlášení, a jednak do souboru, kde jsou uloženi všichni registrovaní uživatelé.

Soubor s registrovanými uživateli obsahuje trojice *jméno-sůl-verifier*. Po přidání nového uživatele se obsah souboru smaže. Následně se vytvoří z registrovaných uživatelů, uložených v seznamu, zašifrovaný obsah, který je pak uložen do souboru. Obsah souboru je vždy při startu serveru načten do paměti (do seznamu), aby byl umožněn rychlý přístup v případě přihlašování klienta k odběru streamu.

Na stránce, kde je formulář pro přidání klientů, je i tlačítko na jejich smazání. Soubor s klienty pak po stisknutí tlačítka bude prázdný. Dále je zde pro větší přehlednost uveden i seznam registrovaných klientů (pouze jména).

Certifikát

Aby bylo možné používat *HTTPS* protokol, musí mít server vlastní certifikát. Pro naše účely byl zvolen pouze *self-signed* certifikát (viz kap. 3.2.1). Tento certifikát je možné vygenerovat pomocí knihovny OpenSSL podle ukázky 4.4. V ukázce byl certifikát vytvořen pod systémem Windows, ale stejný postup by se aplikoval i pod systémy GNU/Linux.

```
1: genrsa -out private.pem 2048
2: req -config C:\path-to\openssl.cnf -x509 -days 3650 -new -key
   private.pem -out public.pem
3: pkcs12 -export -in public.pem -inkey private.pem -out mycert.pfx
```

(4.4)

Na prvním řádku je vygenerován RSA privátní klíč o délce 2048 bitů. Na druhém řádku je vytvořen certifikát *public.pem* ve formátu *x509* s expirační dobou 10 let od data vytvoření. Tento certifikát se bude webovým prohlížečům jevit jako nedůvěryhodný. Tomu se lze vyhnout instalací certifikátu do systému nebo prohlížečů. Instalační certifikát se v příslušném formátu vytvoří pomocí třetího řádku. Server pak používá soubory *public.pem* a *private.pem*.

4.4.2 Ukládání hodnot

Ve webovém rozhraní je i kromě přidávání klientů možné měnit i nastavení serveru, parametrů streamu a parametrů pro detekci změn. Všechny tyto hodnoty jsou uchovávány v souborech (*avoption, avset, server*). Tyto soubory nejsou nijak šifrovány. Kromě těchto souborů jsou na serveru ještě ty samé soubory s příponou *_o*, které uchovávají originální nastavení všech nastavitelných hodnot. Tyto hodnoty je možné opět nastavit pomocí tlačítka *Original settings* u příslušných formulářů pro změnu hodnot parametrů. Po stisku se hodnoty z originálního souboru překopírují do příslušného souboru s nastavením.

Na stránce pro změnu parametrů streamu je textové pole, do kterého je možné zadat celý příkaz pro FFmpeg, který se pak beze změn spustí při dalším vyžádání streamu. Tato funkcionalita byla vytvořena pro zkušené uživatele, kteří vědí, co dělají. Uložený příkaz se bude zapínat do té doby, než dojde k restartování serveru. Pokud se uloží prázdné pole místo příkazu, bude se pro získávání streamu opět používat ten napevno zadaný.

Změna nastavení portů serveru je dostupná až po restartu serveru. Stejně tak změny parametrů streamu jsou aplikovány až na další spuštění nástroje *ffmpeg.exe*, jak bylo již několikrát zmiňováno.

4.5 Grafické rozhraní

Kvůli jednoduššímu ovládání pro uživatele bylo na straně klienta vytvořeno grafické rozhraní. Pro grafický vzhled přehrávače byly použity Qt knihovny (více [24]). Uživateli se nejdříve zobrazí přihlašovací okno (viz kap. 4.2.1 a obr. P 1), a pak okno přehrávače (viz obr. P 2) obsluhované třídou *Main_window*. Jednotlivá tlačítka přehrávače jsou mezi sebou svázána pomocí signálů, které poskytuje Qt. Díky tomu je možné znepřístupnit uživateli některé nesmyslné kombinace, které by mohl zadat (např. nechat si posílat stream jen při změnách, ale už se nepřihlásit k žádnému odběru změn). Stejně tak je po stisku zablokováno tlačítko *Play* a odblokováno je až po tom, co se stiskne *Stop*. Tato funkcionalita zabraňuje uživateli vyžádání dalšího streamu, dokud ten předchozí neukončí.

Podle toho, co uživatel zaškrtá na panelu přehrávače, se pak spustí stream. Pokud se má stream začít přehrávat hned, na server se pošle jak zpráva s žádostí o šifrovací klíč, tak zpráva se zařazením klienta do seznamu, kterému je stream posílán. V případě zapínání streamu až při změně, se pošle pouze zpráva s žádostí o klíč a zpráva s portem, kde poslouchá klientský UDP server, se pošle až když klient obdrží zprávu o změně obrazu/hlasitosti. V případě zapínání přehrávání streamu až při změně je spuštěn odpočet (v samostatném vlákně) a pokud nepřijde další zpráva indikující změnu, než odpočet skončí, okno se streamem se zavře. Jinak se odpočet nastavení opět na výchozí hodnotu a začíná se odpočítávat znovu.

Přehrávač umožňuje zobrazování zpráv o změnách v oznamovací oblasti (system tray) u své ikony (viz obr. P 4). Aby se třída *Main_window* dozvěděla, že byla přijata zpráva indikující změnu, musí být zaregistrována k odběru oznámení pomocí signálů. Signály jsou generovány třídou *Communication*, která spravuje příjem zpráv od serveru. Stejně jako na serveru, i zde byl použit návrhový vzor *Observer*.

Kromě signálů o změnách se zde posílají i signály indikující ukončení serveru (nebo chybu přenosu). Pokud je server ukončen, je potřeba vypnout přehrávač, aby klient zbytečně nečekal na něco, co nenastane. V případě selhání funkce, která přijímá zprávy od serveru, nebo obdržení ukončovací zprávy serveru, se nejdříve provede ukončení všech běžících vláken, a nakonec se zobrazí dialogové okno, které informuje o ukončení serveru a ukončení aplikace. Po objevení tohoto okna již není možné cokoliv na přehrávači zadat a je možné ho pouze ukončit.

4.6 Konzolové rozhraní

Server byl vytvořen jako konzolová aplikace. Nastavení serveru je možné spravovat přes webové rozhraní, a přes konzoli je možné ho, zadáním příkazu *quit*, legálně ukončit. Toto ukončení bylo implementováno proto, aby všechna běžící vlákna mohla být regulérně ukončena. Kvůli použitým timeoutům na funkcích *select()*, není ukončení serveru okamžité, ale trvá pár sekund. Kromě vláken obsluhujících klienty a UDP servery se musí ukončit i externí programy *ffmpeg.exe* a *mdns_server.exe*.

Do konzole, kde běží server, je vypisován log. Tento log je určen spíše programátorům než běžným uživatelům aplikace. Log se kromě konzole ukládá i do souboru

LOG_SERVER, kam je stále přidáván. Logování je možné programově vypnout ve třídě *Log*, kde stačí změnit direktivu *#define* na *#undef* u definovaných maker *LOG_FILE* a *LOG_TERMINAL*.

Stejný princip logování je vytvořen i v programu klienta s tím rozdílem, že log se zapisuje pouze do souboru.

5 Naměřené hodnoty

V této kapitole budou uvedeny hodnoty, které byly změřeny během testování aplikace. Webové rozhraní bylo otestováno v prohlížečích Opera 12.16, Mozilla Firefox 28.0, Internet Explorer 11 a Google Chrome 33.0.1750.154 m.

5.1 Využití šířky pásma

Důležitým faktorem je využití šířky pásma. Aplikace byla spuštěna několika způsoby, a vždy byly zaznamenány hodnoty odesílaných a přijímaných dat v *B/s* u jednotlivých částí aplikace. Výsledky jsou uvedeny v tabulce 5.1. Pro nastavení streamu byly použity výchozí hodnoty (rozlišení = 800x600, bit rate = 282 000 b/s, mono, audio frekvence = 16 000*Hz*, *počet rámců/s* = 25, kvalita = 9) a nebylo vyžádáno posílání zpráv při změně. *DP-server.exe* je program serveru a *DP-client.exe* je program klienta.

Z naměřených hodnot je vidět, že typ přenášeného streamu nemá žádný vliv na množství přenesených dat z *ffmpeg.exe*, jelikož se spouští stále stejný příkaz, ve kterém jsou 2 výstupní streamy. Dále je vidět pokles množství přenášených dat, pokud se místo audia+videa přenáší pouze audio. Zajímavé srovnání poskytují řádky 5 a 6, kde na řádku 5 byl puštěn stream s nejvyšší kvalitou a na řádku 6 naopak s nejnižší. Objem odeslaných dat je u všech programů o více než polovinu menší. Kromě *ffmpeg.exe*, zde došlo pouze k cca třetinovému snížení.

5.2 Využití paměti

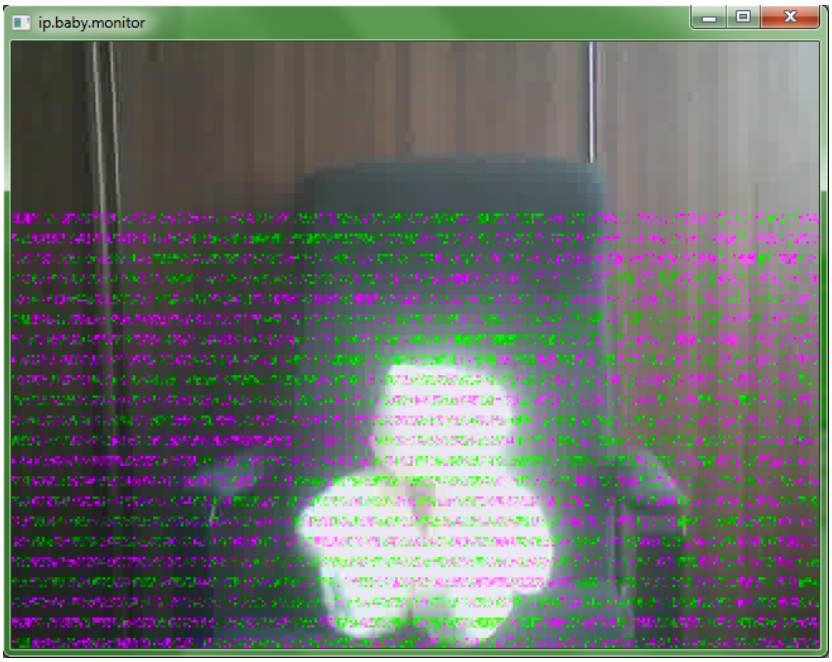
Dalším faktorem, který je měřitelný, je využití operační paměti jednotlivými částmi aplikace (viz tabulka 5.2). Tato měření probíhala ve stejnou dobu jako výše zmíně měření přenášených dat. Z naměřených hodnot jsou vidět větší požadavky na paměť v případě přehrávání audia+videa pomocí *ffplay.exe*. Na řádcích 5 a 6 jsou obrovské rozdíly, co se týče využití paměti nástrojem *ffmpeg.exe*, který při nastavení nejvyšší kvality videa spotřebovává 6x více paměti než při nejnižší kvalitě. Stejně tak program *ffplay.exe* potřebuje pro kvalitnější stream mnohem více paměti.

5.3 Přehrávání streamu

Stream, který je na straně klienta zobrazován, je asi o 3 vteřiny pozadu oproti realitě. Menší prodlevy se podařilo dosáhnout použitím parametru *-tune zerolatency* u získávání streamu, ale pak docházelo k nežádoucímu rušení obrazu v podobě barevných proužků v dolní části obrazovky (viz obr. 5.1) nebo k úplnému rozhození obrazu, že pak nebylo vůbec poznat, co se snímá. Na straně klienta se příchozí data nebufferují, což snížilo prodlevu oproti realitě o asi 10 sekund.

Přesto, že dochází k občasnému malému rušení v dolní části obrazu, je jeho kvalita dobrá. Rušení zmizí, pokud se před kamerou mávne (dojde tím k velké změně a FFmpeg pošle celý nový I-rámec). Bylo zjištěno, že rušení obrazu je menší při použití 30 *snímků/s* místo 25, které jsou nastaveny jako výchozí hodnota. Zvuk s obrazem je synchronizován.

Při testování aplikace došlo k tomu, že některá zařízení (Windows 7 Professional, 64 b) nepropustila přes svůj Firewall data streamu. Nepomohlo ani zapnutí přenosu pouze audio streamu. Po vypnutí Firewallu aplikace normálně fungovala. Firewall byl pouze systémový.



Obr. 5.1: Nežádoucí rušení obrazu.

#	Spuštění	Program	Příjem $[B/s]$	Odeslání $[B/s]$
1	Klient si vyžádal souvislé přehrávání streamu (video + audio).	DP-server.exe	51 196	36 771
		DP-client.exe	36 771	36 237
		ffmpeg.exe	0	66 777
		ffplay.exe	48 316	0
		mdns_server.exe	5	5
2	Klient si vyžádal souvislé přehrávání streamu (audio).	DP-server.exe	59 164	17 681
		DP-client.exe	17 681	17 460
		ffmpeg.exe	0	64 125
		ffplay.exe	18 749	0
		mdns_server.exe	5	5
3	Klient si vyžádal souvislé přehrávání streamu (video + audio) a ukládání streamu do souboru.	DP-server.exe	60 260	42 911
		DP-client.exe	42 911	84 703
		ffmpeg.exe	0	62 533
		ffplay.exe	44 795	0
		mdns_server.exe	5	5
		ffmpeg_save_file.exe	44 778	0
4	Klient si vyžádal souvislé přehrávání streamu (audio) a ukládání streamu do souboru.	DP-server.exe	48 577	14 785
		DP-client.exe	14 785	29 211
		ffmpeg.exe	0	63 762
		ffplay.exe	19 273	0
		mdns_server.exe	5	5
		ffmpeg_save_file.exe	19 273	0
5	Klient si vyžádal souvislé přehrávání streamu (video + audio) v nejvyšší kvalitě.	DP-server.exe	30 809	22 449
		DP-client.exe	38 319	37 187
		ffmpeg.exe	0	68 390
		ffplay.exe	50 047	0
6	Klient si vyžádal souvislé přehrávání streamu (video + audio) v nejnižší kvalitě.	DP-server.exe	10 890	6 219
		DP-client.exe	16 204	15 978
		ffmpeg.exe	0	41 837
		ffplay.exe	23 939	0

Tab. 5.1: Využití šířky pásma programy.

#	Spuštění	Program	Využití paměti [kB]
1	Klient si vyžádal souvislé přehrávání streamu (video + audio).	DP-server.exe	2 132
		DP-client.exe	8 100
		ffmpeg.exe	147 516
		ffplay.exe	35 616
		mdns_server.exe	1 412
2	Klient si vyžádal souvislé přehrávání streamu (audio).	DP-server.exe	7 832
		DP-client.exe	2 196
		ffmpeg.exe	146 489
		ffplay.exe	8 196
		mdns_server.exe	1 352
3	Klient si vyžádal souvislé přehrávání streamu (video + audio) a ukládání streamu do souboru.	DP-server.exe	2 200
		DP-client.exe	7 680
		ffmpeg.exe	145 500
		ffplay.exe	31 576
		mdns_server.exe	1 352
		ffmpeg_save_file.exe	38 064
4	Klient si vyžádal souvislé přehrávání streamu (audio) a ukládání streamu do souboru.	DP-server.exe	2 172
		DP-client.exe	7 668
		ffmpeg.exe	141 952
		ffplay.exe	7 188
		mdns_server.exe	1 448
		ffmpeg_save_file.exe	3 620
5	Klient si vyžádal souvislé přehrávání streamu (video + audio) v nejvyšší kvalitě.	DP-server.exe	2 376
		DP-client.exe	7 620
		ffmpeg.exe	440 384
		ffplay.exe	122 568
6	Klient si vyžádal souvislé přehrávání streamu (video + audio) v nejnižší kvalitě.	DP-server.exe	2 544
		DP-client.exe	7 684
		ffmpeg.exe	70 328
		ffplay.exe	17 164

Tab. 5.2: Využití paměti programů.

6 Závěr

Přestože je v dnešní době na trhu dostupné nepřeberné množství dětských chůviček, ne všechny jsou použitelné, ať už proto, že se ruší s WiFi sítí nebo proto, že neposkytují šifrovaný přenos dat, který je důležitý pro bezpečnost a ochranu soukromí jejich uživatelů. Tato práce se tedy zabývá analýzou současných řešení chůviček, jejich nedostatků i předností. Na základě této analýze jsou pak stanoveny požadavky na novou chůvičku, která má za úkol odstranit 2 hlavní nedostatky, a to rušení s WiFi sítí a šifrování přenosu (audia, videa).

Byly tak vytvořeny dva programy, které ke své vzájemné komunikaci využívají již zavedené internetové spojení. Jedním z těchto programů je server, který musí běžet na zařízení, které obsahuje kameru a mikrofon, a je tedy určen ke sledování dítěte. Naopak druhý program - klient musí být spuštěn na zařízení s monitorem a reproduktory a slouží jako druhá část chůvičky, kterou u sebe mají rodiče.

Na klientovi je tedy možné sledovat buď audio nebo audio+video stream, který je odchytáván v pokoji dítěte. Kromě toho je možné zapnout i detekci pohybu a hlasitosti. Když začne dítě brečet nebo se probudí a začne se snažit dostat z postýlky, rodič o tom dostane zprávu, která se po určitou dobu zobrazuje v oznamovací oblasti. Pokud se stane, že se po dobu zobrazení zprávy rodič zrovna nesledoval monitor, pak je možné si nechat poslední událost zobrazit kliknutím na ikonu klienta. Přehrávaný stream je možné ukládat do souboru.

Oba programy (klient, server) spolu komunikují pomocí šifrovaného spojení, a též stream, který je posílán ze serveru klientovi, je zašifrovaný. Pro snadnější nastavení serveru bylo implementováno webové rozhraní, kde je možné nastavit jak parametry serveru (porty), tak i parametry přenášeného streamu (rozlišení, kvalita videa atd.). Toto nastavení je společné pro všechny klienty, kteří se na server připojí a odebírají stream. Každý klient zvlášť si ale může vybrat, zda chce sledovat audio, audio+video, o jakých změnách bude informován, a zda se bude stream přehrávat hned od začátku nebo jen po určitou dobu při každé detekci změny.

Práce splňuje všechny body zadání, které byly stanoveny. Kromě již implementovaných součástí by bylo možné práci rozšířit o další funkcionality, které by server/klient nabízel (např. zobrazování streamu z více kamer najednou).

Literatura

[1] A forum dedicated to FFmpeg on Windows. In: *Zeranoe FFmpeg* [online]. 2013 [cit. 2014-04-26]. Dostupné z : <http://ffmpeg.zeranoe.com/-forum/viewtopic.php?f=5&t=1309>.

[2] An MPEG frame sequence with two possible references: a P-frame referring to a I-frame and a B-frame referring to two P-frames. *Video Compression* [online]. MITROVIC, Djordje.. 2006 [cit. 2014-04-15]. Dostupné z : <http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV-0506/s0561282.pdf>.

[3] ANDERS, Jörg. *MPEG video compression technique* [online]. 2007 [cit. 2014-04-18]. Dostupné z : <http://vsr.informatik.tu-chemnitz.de/~jan/MPEG/HTML/mpeg_tech.html>.

[4] Announcing the ADVANCED ENCRYPTION STANDARD. In: *Federal Information Processing Standards Publication 197* [online]. 2001 [cit. 2014-04-15]. Dostupné z : <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[5] ARORA, Mohit. How secure is AES against brute force attacks?. In: *Design How-To* [online]. 2012 [cit. 2014-04-17]. Dostupné z : <http://www.eetimes.com/document.asp?doc_id=1279619>.

[6] C++ - Check an IP Address is in a IP/Mask range. In: *Random Stuff from a software developer* [online]. 2012 [cit. 2014-04-21]. Dostupné z : <http://www.stev.org/post/2012/08/09/C++-Check-an-IP-Address-is-in-a-IPMask-range.aspx>.

[7] Demonstration of visual cryptanalysis of the Data Encryption Standard's ECB mode. *Security data visualization*. CONTI, Greg. San Francisco: Starch Press, Inc. 2007. ISBN-13: 978-1-59327-143-5.

[8] Depending on the subsampling, 2 or 4 pixel values of the chrominance channel can be grouped together. *Video Compression* [online]. MITROVIC, Djordje.. 2006 [cit. 2014-04-15]. Dostupné z : <http://homepages.inf.ed.ac.uk/-rbf/CVonline/LOCAL_COPIES/AV0506/s0561282.pdf>.

[9] DOSTÁLEK, Libor a Alena KABELOVÁ. *Velký průvodce protokoly TCP/IP a systémem DNS* Praha: Computer Press, 2000. ISBN 80-7226-323-4.

[10] DWORKIN, Morris. Recommendation for Block Cipher Modes of Operation. In: *NIST Special Publication 800-38A* [online]. 2001 [cit. 2014-04-15]. Dostupné z : <http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>.

[11] FAIRHURST, Gorz. MPEG-2 Transmission. In: *MPEG-2 Digital Video* [online]. 2001 [cit. 2014-04-17]. Dostupné z : <http://www.erg.abdn.ac.uk/future-net/digital-video/mpeg2-trans.html>.

[12] *FFmpeg* [online]. 2014 [cit. 2014-04-17]. Dostupné z : <http://www.ffmpeg.-org/>.

[13] FORRET, Peter. Video bitrate calculator. In: *Web.forret.com* [online]. 2014 [cit. 2014-04-18]. Dostupné z : <http://web.forret.com/tools/video_fps.-asp>.

[14] FREIER, A. a P. KARLTON. The Secure Sockets Layer (SSL) Protocol Version 3.0. In: *IETF Documents* [online]. 2011 [cit. 2014-04-13]. ISSN 2070-1721. Dostupné z : <http://tools.ietf.org/html/rfc6101#section-5.6.2> .

[15] CHANDRA, Pravir, Matt MESSIER a John VIEGA. *Network Security with OpenSSL.* O'Reilly, 2002. ISBN 0-596-00270-X.

[16] CHESHIRE, S. KROCHMAL, M. Multicast DNS. In: *Internet Engineering Task Force (IETF)* [online]. 2013 [cit. 2014-04-15]. Dostupné z : <http://tools.ietf.org/html/rfc6762#page-4>.

[17] KERN, Christoph, Anita KESAVAN a Neil DASWANI. *Foundations of Security: What Every Programmer Needs to Know.* New York: Springer-Verlag New York, 2007. ISBN 978-1590597842.

[18] Levels of Noise In decibels (dB). In: *American Academy of Audiology* [online]. 2009 [cit. 2014-04-15]. Dostupné z : <http://www.audiology.org/practice/resources/PublishingImages/NoiseChart16x20.pdf>.

[19] MITROVIC, Djordje. *Video Compression* [online]. 2006 [cit. 2014-04-15]. Dostupné z : <http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0506/s0561282.pdf>.

[20] MP3 (MPEG Layer III Audio Encoding). *Sustainability of Digital Formats* [online]. 2012 [cit. 2014-04-18]. Dostupné z : <http://www.digitalpreservation.gov/formats/fdd/fdd000012.shtml>.

[21] *OpenSSL: The Open Source toolkit for SSL/TLS* [online]. 2014 [cit. 2014-04-18]. Dostupné z : <https://www.openssl.org/>.

[22] POSIX Threads for Win32. In: *Pthreads w32* [online]. 2012 [cit. 2014-04-15]. Dostupné z : <http://www.sourceware.org/pthreads-win32/>.

[23] Pseudo Code for Key Expansion. In: *Announcing the ADVANCED ENCRYPTION STANDARD* [online]. Federal Information Processing Standards Publications . 2001 [cit. 2014-04-15]. Dostupné z : <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[24] *Qt Project* [online]. 2014 [cit. 2014-04-24]. Dostupné z : <http://qt-project.org/>.

[25] RESCORLA, E. HTTP Over TLS. *IETF Documents* [online]. 2000 [cit. 2014-04-13]. Dostupné z : <https://tools.ietf.org/html/rfc2818>.

[26] ShiftRows() cyclically shifts the last three rows in the State. In: *Announcing the ADVANCED ENCRYPTION STANDARD* [online]. Federal Information Processing Standards Publications . 2001 [cit. 2014-04-15]. Dostupné z : <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[27] STEVENS, Marc. Single-block collision attack on MD5. In: *Cryptology Group, CWI* [online]. 2011 [cit. 2014-04-15]. Dostupné z : <http://marc-stevens.nl/research/md5-1block-collision/md5-1block-collision.pdf>.

[28] SubBytes() applies the S-box to each byte of the State. In: *Announcing the ADVANCED ENCRYPTION STANDARD* [online]. Federal Information Processing Standards Publications . 2001 [cit. 2014-04-15]. Dostupné z : <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[29] ŠLECHTA, Pavel. *C++ library implementing The Stanford Secure Remote Password Protocol - SRP (SRP6a)* [online]. 2012 [cit. 2014-04-21]. Dostupné z : <https://github.com/slechta/DragonSRP>.

[30] The CBC Mode. In: *NIST Special Publication 800-38A* [online]. DWORKIN, Morris. 2001 [cit. 2014-04-15]. Dostupné z : <http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>.

[31] The CFB Mode. In: *NIST Special Publication 800-38A* [online]. DWORKIN, Morris. 2001 [cit. 2014-04-15]. Dostupné z : <http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>.

[32] TAN, Darell. Publishing Services over mDNS in C In: *Adventures of an electronics & software guy* [online]. 1997 [cit. 2014-04-15]. Dostupné z : <http://irq5.io/2011/04/10/publishing-services-over-mdns-in-c/>.

[33] TANENBAUM, ANDREW S. a DAVID J. WETHERALL. *Computer Networks.* 5. vyd. New Jersey: Prentice Hall, 2010. ISBN 978-0132126953.

[34] WANG, Xiaoyun. YU, Hongbo. How to break MD5 and other hash functions. *Advances in Cryptology* [online]. 2005 [cit. 2014-04-15]. Dostupné z : <http://download.springer.com/static/pdf/55/chp%253A10.1007%252F1-1426639_2.pdf?auth66=1397767649_feebb39e5ed797254410f1a5628dbd53&-ext=.pdf>.

[35] wiki: Creating multiple outputs. In: *FFmpeg* [online]. 2014 [cit. 2014-04-19]. Dostupné z : <http://trac.ffmpeg.org/wiki/Creating%20multiple%20outputs>.

[36] WU, Thomas. SRP-6: Improvements and Refinements to the Secure Remote Password Protocol. In: *Submission to the IEEE P1363 Working Group* [online]. 2002 [cit. 2014-04-15]. Dostupné z : <http://srp.stanford.edu/srp6.-ps>.

[37] WU, Thomas. The secure remote password protocol. In: *Internet Society Network and Distributed System Security Symposium* [online]. 1997 [cit. 2014-04-15]. Dostupné z : <http://srp.stanford.edu/ndss.html>.

Přílohy

Uživatelská příručka

Aplikace se skládá ze dvou částí, serveru a klienta. Server musí být spuštěn na zařízení, na kterém je kamera a mikrofon (u dítěte), a klient na zařízení s monitorem a reproduktory (u rodičů). Aplikace byla primárně vytvořena pod systémem Windows 7, ale je ji možné přeložit i pod systémem Debian (pokud je na něm nainstalováno Qt a OpenSSL). Pod Windows je nutné mít nainstalován *DirectShow* (pro překlad aplikace i Qt a OpenSSL).

Instalace

Pod systémy Windows je možné použít instalační balíček, který provede nainstalování programu. Při instalaci jsou překopírovány všechny potřebné soubory do složky, kterou si uživatel vybere. Po nainstalování je ještě možné vytvořit zástupce, a toho umístit např. na plochu. Pro server a klienta je tento postup trochu odlišný. Instalační soubory se nachází ve složkách *install* pro server i klienta. Pokud se nedodrží pokyny níže, programy sice půjdou spustit, ale nebudou fungovat všechny jejich součásti.

Server

Postup pro nainstalování serveru je následující:

1. Spustit soubor *server/install/server_setup.exe*.
2. Postupovat podle pokynů na obrazovce.

3. Po dokončení instalace je možné vytvořit zástupce na ploše.
4. Přemístěte se do složky, kam se program nainstaloval, a pravým tlačítkem klikněte na soubor *DP-server.exe* a zvolte *Odeslat → Plocha(vytvořit zástupce)*.
5. Pokud byl program nainstalován do *Program Files*, pak je nutné spouštět tohoto zástupce vždy jako administrátor.
6. Pokud byl program nainstalován jinam, je možné ho spouštět normálně.

Klient

Postup pro nainstalování klienta je následující:

1. Spustit soubor *klient/install/client_setup.exe*.
2. Postupovat podle pokynů na obrazovce.
3. Po dokončení instalace je možné vytvořit zástupce na ploše.
4. Přemístěte se do složky, kam se program nainstaloval, a pravým tlačítkem klikněte na soubor *DP-server.exe* a zvolte *Odeslat → Plocha(vytvořit zástupce)*.
5. Není nutné spouštět tento program jako administrátor.

Spuštění

Programy lze spustit buď přímo ze složky, kde jsou nainstalované *DP-server.exe* a *DP-client.exe*, nebo pomocí zástupce, který byl vytvořen podle návodu v předešlé kapitole. Po spuštění (serveru nebo klienta) budete vyzváni k povolení využívání síťového připojení. Síťové připojení je pro chod obou programů nezbytné.

Server

Po kliknutí na ikonu serveru se otevře konzole, a v ní by se měl objevit výpis podobný ukázce 6.1. To, co se bude lišit, je 2. a 3. řádek, protože zde se vypisují

dostupná zařízení pro snímání obrazu (*1.3M HD WebCam*) a zvuku (*Mikrofon (Conexant High Definit*).

```
16:04:35-25.04.2014 Multicast server has started on port 9999
16:04:36-25.04.2014 FFmpeg: Found video device: "1.3M HD WebCam"
16:04:36-25.04.2014 FFmpeg: Found audio device: "Mikrofon (Conexant
High Definit"
16:04:36-25.04.2014 HTTPS Server has started on port 9443
16:04:36-25.04.2014 File src/client/files/clients was decrypted.
16:04:36-25.04.2014 Users have been loaded from file.
16:04:36-25.04.2014 Server has started on port 9001
```

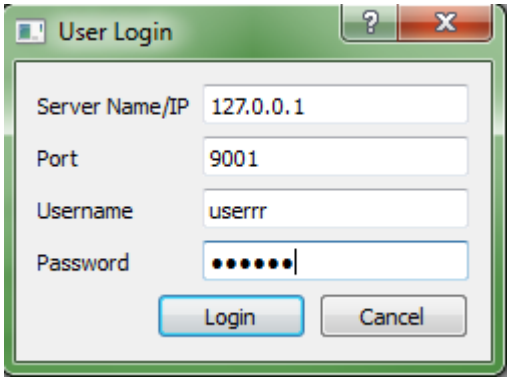
(6.1)

To, co se bude dále v konzoli zobrazovat, není důležité, jedná se o logovací výpis událostí na serveru. Server je možné ukončit zavřením konzole. Mnohem lepší je ale do konzole napsat slovo *quit*. Tímto dojde k bezpečnému ukončení serveru. Server se ukončí v okamžiku, kdy se konzole zavře. Pokud je stále otevřená, mohlo se stát, že se jednu součást serveru nepodařilo vypnout, jedná se o program *mdns_server.exe* a je nutné ho ukončit ručně přes *Správce úloh systému Windows*.

Klient

Klient se spouští až po tom, co server běží. Po spuštění se objeví přihlašovací okno (viz obr. P 1). V tomto okně se vyplňuje IP adresa serveru nebo název serveru, který je nastaven na *ip.baby.monitor*. Druhé pole obsahuje port, na kterém server běží, a pokud se neměnilo nastavení, měl by odpovídat číslu *9001*. Poslední dvě pole jsou uživatelské jméno a heslo. Tyto údaje je nejdříve nutné registrovat pomocí webového rozhraní (viz kap. 6).

Po stisku tlačítka *Login* se spustí přihlašovací proces, který může skončit hned několika chybami:



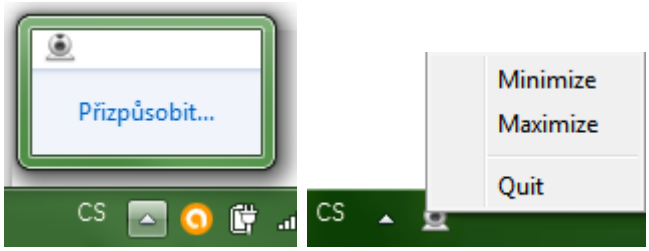
Obr. P 1: Přihlašovací okno do aplikace klienta.

- `Connection to server failed.` - Spojení se serverem nebylo navázáno. Buď je špatně IP adresa, port nebo server vůbec není zapnutý.
- `Authentication failed. Username or password is incorrect.` - Pokud je zadáno špatné uživatelské jméno nebo heslo.
- `Username can not be empty.` - Pokud se nezadá uživatelské jméno.
- `Password can not be empty.` - Pokud se nezadá heslo.
- `Server port is not valid (9000-65535).` - Číslo portu neodpovídá povolenému rozsahu.

Pokud vše proběhne tak, jak má, otevře se okno přehrávače (viz obr. P 2) a v oznamovací oblasti se vytvoří ikona (malá web kamera) pro tento přehrávač (viz obr. P 3 (a)). Po úspěšném spojení se serverem se IP adresa uloží do souboru. Po stisku ikony pravým tlačítkem myši se objeví menu (viz P 3 (b)), které umožňuje zvětšení/zmenšení a ukončení přehrávače. Okno přehrávače je rozděleno mezerou na dvě části, kde levá slouží pro nastavení streamu a pravá pro jeho spuštění/ukončení/ukládání. Pokud chce uživatel sledovat video+audio nebo jen audio, vždy musí stisknout tlačítko *Play*. Pokud chce změnit nastavení a sledovat např. místo videa jen audio, pak je vždy nutné přehrávání nejdříve ukončit pomocí *Stop*, změnit nastavení a znovu zapnout.

Levá část je rozdělena do menších sekcí. První s názvem *Stream* umožňuje vybrat, zda se bude přehrávat video+audio nebo jen audio. Pokud se zaškrtne samotné audio, pak je možné zrušit zobrazování okno (*Display window*), ve kterém se, za jiných okolností, zobrazuje video. V případě audia v tomto okně běží zvuková vlna.

Obr. P 2: Klientský program - přehrávač.

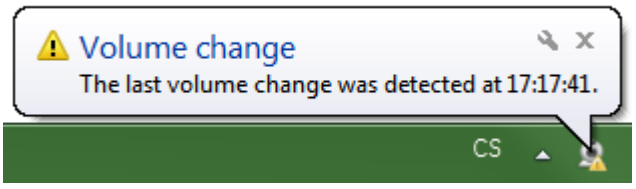


(a) Ikona (b) Menu

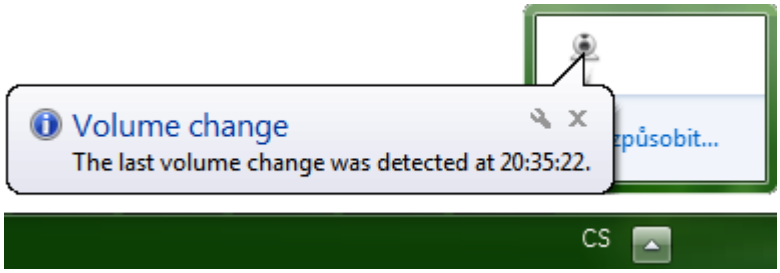
Obr. P 3: Ikona přehrávače.

Vypnutí zobrazovacího okna má vliv pouze na audio stream, u video streamu vypnout nelze.

Druhá sekce s názvem *Change monitoring* dává vybrat, zda chce být uživatel upozorněn, pokud nastane zvýšení hlasitosti (*Audio change*) nebo pohyb před kamerou (*Video change*). Pokud je něco zaškrtnuto, pak pokaždé, když příslušná změna nastane, obdrží uživatel varování v podobě zobrazení zprávy v oznamovací oblasti (viz obr. P 4). Kromě toho se změní i ikona přehrávače, přibude u ní malý žlutý vykřičník. Pokud se na ikonu přehrávače klikne, když je u ní tento vykřičník, pak se zobrazí poslední změna, která nastala (viz obr. P 5).



Obr. P 4: Zobrazení zprávy u klienta při změně.



Obr. P 5: Zobrazení poslední obdržené zprávy o změně v oznamovací oblasti po stisku ikony přehrávače.

Poslední sekce má název *Turn on* a dává uživateli na výběr, zda chce zapnout přehrávání hned nebo až při změně. Pokud se vybere přehrávání streamu hned (*Play now*), pak je po stisku tlačítka *Play*, asi po 10 sekundách, spuštěno okno, ve kterém běží stream odchytávaný kamerou na serveru. Při výběru *Play on change* se zobrazovací okno zapne až v okamžiku(opět po 10 sekundách), kdy se objeví v oznamovací oblasti zpráva o změně. Okno pak zůstane otevřené 15 sekund, a pak se zavře. Doba, jak dlouho má být okno otevřené, se dá nastavit v konfiguračním souboru (viz 6).

Pokud je zaškrtnuté přehrávání hned (*Play now*), pak je možné ukládat přehrávaný stream do souboru. Při přehrávání při změně to možné není. Pro ukládání do souboru musí být zaškrtnuté políčko s nápisem (*Save to file*). Pomocí tlačítka (*Browse*) je možné si vybrat, kam se soubor uloží. Název souboru je dán napevno a obsahuje datum a čas, kdy se soubor začal ukládat.

Pokud na obou zařízeních běží programy *ffmpeg.exe* i *ffplay.exe* a přesto se žádný stream nepřehrává, může to být způsobeno tím, že příchozí data streamu blokuje firewall.

Webové rozhraní

Nastavení serveru je přístupné přes webové rozhraní. Toto rozhraní se dá otevřít v běžném webovém prohlížeči na adrese *https://<IP serveru>:9443/*. IP adresa serveru je stejná, jako se vyplňuje pro přihlášení do klientské aplikace. Na této adrese se nachází přihlašovací formulář do rozhraní. Uživatelské jméno je nastaveno na *admin* a heslo na *111111*. Heslo se doporučuje změnit. Veškeré hodnoty, které se ve webovém rozhraní nastavují, se týkají všech klientů.

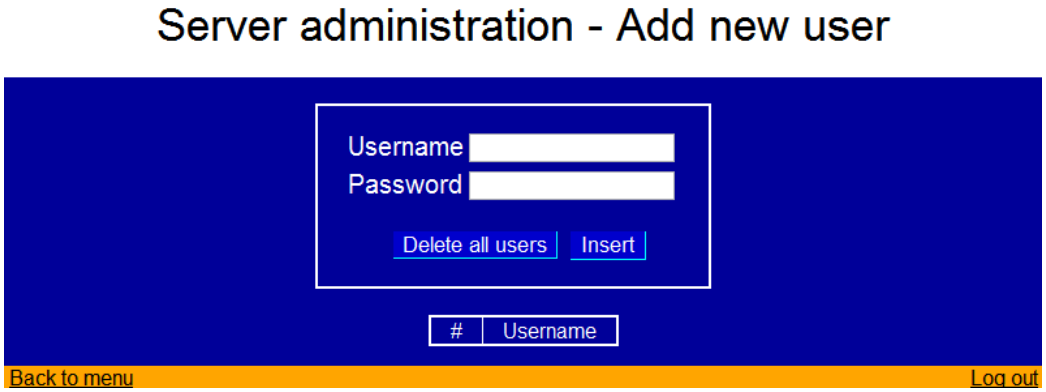
Po úspěšném přihlášení se zobrazí stránka s menu, kde jsou položky:

- `Add user` - Přidávání nových uživatelů, kteří se pak pod těmito údaji mohou přihlásit do klientské aplikace.
- `Audio/Video options` - Nastavení mezí pro detekci změn hlasitosti a pohybu.
- `FFmpeg options` - Nastavení parametrů streamu.
- `Server settings` - Nastavení serveru.

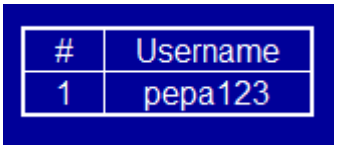
Na každé stránce se opakují dva odkazy, jeden je *Back to menu*, který vrací stránku s hlavním menu, a *Log out*, který provede odhlášení z webového rozhraní.

Přidání nových uživatelů

Stránka obsahuje formulář, kde se vyplní uživatelské jméno a heslo. Jméno i heslo mají stejná pravidla. Oboje může být maximálně 50 a minimálně 6 znaků dlouhé a smí obsahovat pouze znaky: velká a malá písmena bez diakritiky, číslice, tečku, hvězdičku, podtržítko a pomlčku. Pokud bude jméno nebo heslo obsahovat nepovolené znaky nebo bude mít špatnou délku, neuloží se. Uživatel je o úspěchu uložení informován větou, která se objeví nad formulářem po vložení údajů. Pokud se údaje uloží, pod formulářem bude vypsáno jméno nově přidaného uživatele (viz obr. P 7).



Obr. P 6: Stránka pro přidání nových klientů.

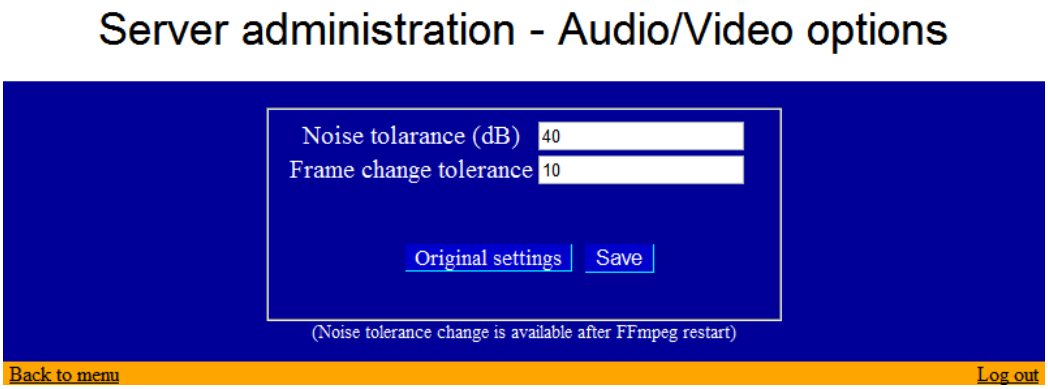


Obr. P 7: Seznam registrovaných uživatelů.

Kromě přidání nových klientů je zde i tlačítko *Delete all users*, které provede smazání všech uživatelů v zobrazeném seznamu. Po této akci se již není možné pod původně zadanými údaji přihlásit.

Nastavení detekce změn

Na této stránce je možné nastavit tolerance pro detekci změn. První pole je pro hlasitost a udává hladinu zvuku v decibelech. Pokud je tato hladina překročena, je poslána zpráva klientovi, která se zobrazí v oznamovací oblasti (viz obr. P 5). Výchozí hodnota je $40dB$, což odpovídá tiché knihovně. Minimální hodnotu lze nastavit na $0dB$ a maximální na $120dB$ *(start tryskového letadla)*. Druhé pole je pro detekci pohybu a hodnoty zobrazuje v procentech. Udává, o kolik procent se musí změnit jeden snímek oproti předchozímu. Výchozí hodnota je 10%.



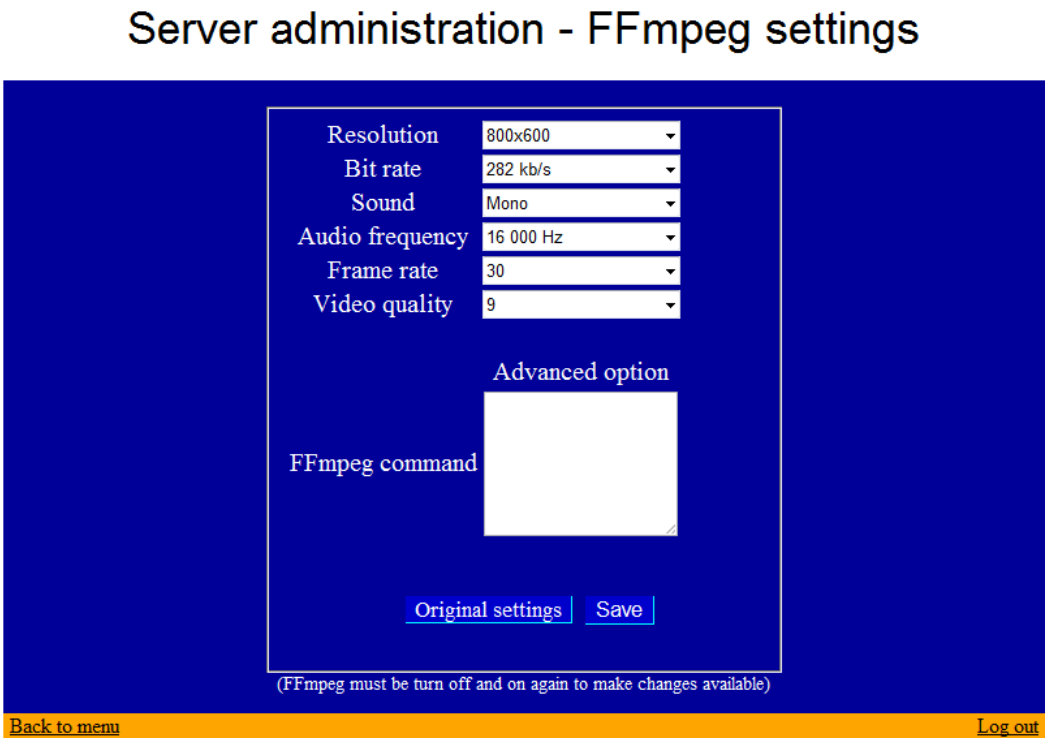
Obr. P 8: Stránka pro nastavení mezí pro detekce změn.

Dále je na této stránce k dispozici tlačítko *Original settings*, které způsobí nastavení výše zmíněných parametrů do výchozích hodnot ($40dB$ a 10%). Změna parametru pro detekci pohybu se projeví ihned i při již zapnutém sledování streamu. Aby se projevila změna tolerance hlasitosti, je nejdříve nutné ukončit přehrávání streamu na všech připojených klientech (není nutné program klienta vypnout úplně). Při dalším zapnutí se již použije nově nastavená hodnota.

Nastavení streamu

Na této stránce je možné upravovat vlastnosti přehrávaného streamu (viz obr. P 9). Veškeré změny se, stejně jako u změny hlasitosti, projeví až po tom, co všichni klienti ukončí přehrávání streamu. Uživatel zde má na výběr vždy z několika hodnot. Parametry, které je možné nastavovat, jsou: rozlišení obrazu, bit rate, počet audio kanálů, audio frekvence, počet snímků za sekundu, kvalita videa.

Kvalita videa je od 1 do 11, kde 1 je nejlepší a 11 nejhorší. Čím vyšší hodnoty se nastaví (kromě kvality), tím bude stream lepší, ale zároveň se musí přenášet větší objem dat a pomalejší připojení by to nemusela zvládat. Výchozí hodnoty jsou uvedeny na obrázku P 9. I zde je možné vrátit změněné hodnoty do původního nastavení tlačítkem *Original settings*.

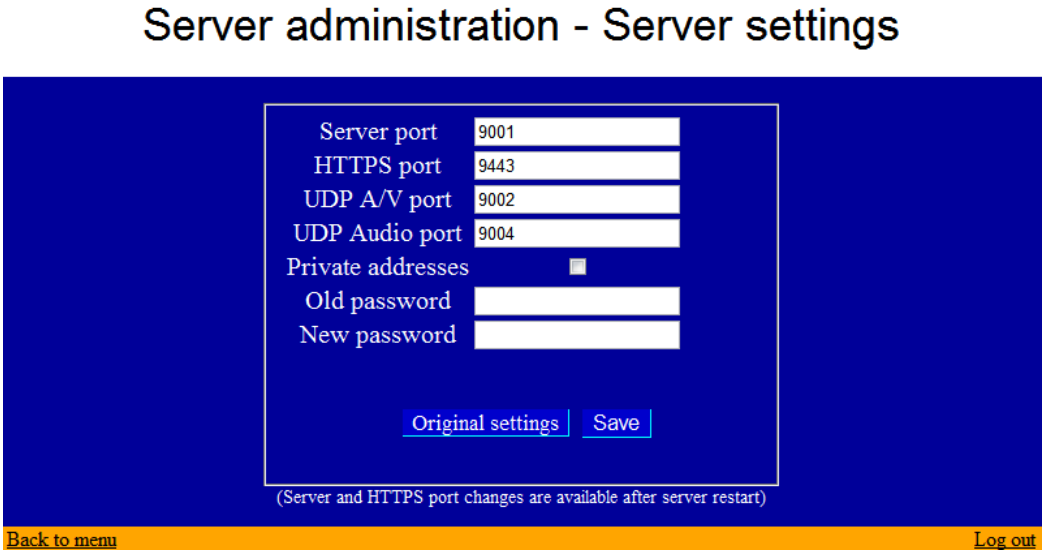


Obr. P 9: Stránka pro nastavení parametrů streamu.

Pole *FFmpeg command* umožňuji zkušeným uživatelům, zadat si svůj vlastní příkaz k získání streamu ze zařízení. Takto uložený příkaz je spouštěn do té doby, než dojde k restartu serveru. Nastavení původního příkazu je možné po uložení prázdného pole *FFmpeg command* .

Nastavení serveru

Na poslední stránce je možné změnit nastavení serveru (viz obr P 10). Toto nastavení se doporučuje neměnit, pokud to není opravdu nutné, tj. porty používané serverem už používá jiná aplikace. Změna portů se samozřejmě projeví až po restartu serveru. Kromě portů je ještě možné měnit heslo do administrátorské části. Pro úspěšnou změnu je nejdříve nutné zadat staré heslo. Nové heslo se řídí stejnými pravidly jako hesla při registraci uživatelů (tj. musí být v rozmezí 6-50 znaků a má omezenou množinu použitelných symbolů). Pokud nechcete měnit heslo, necháte obě pole prázdná.



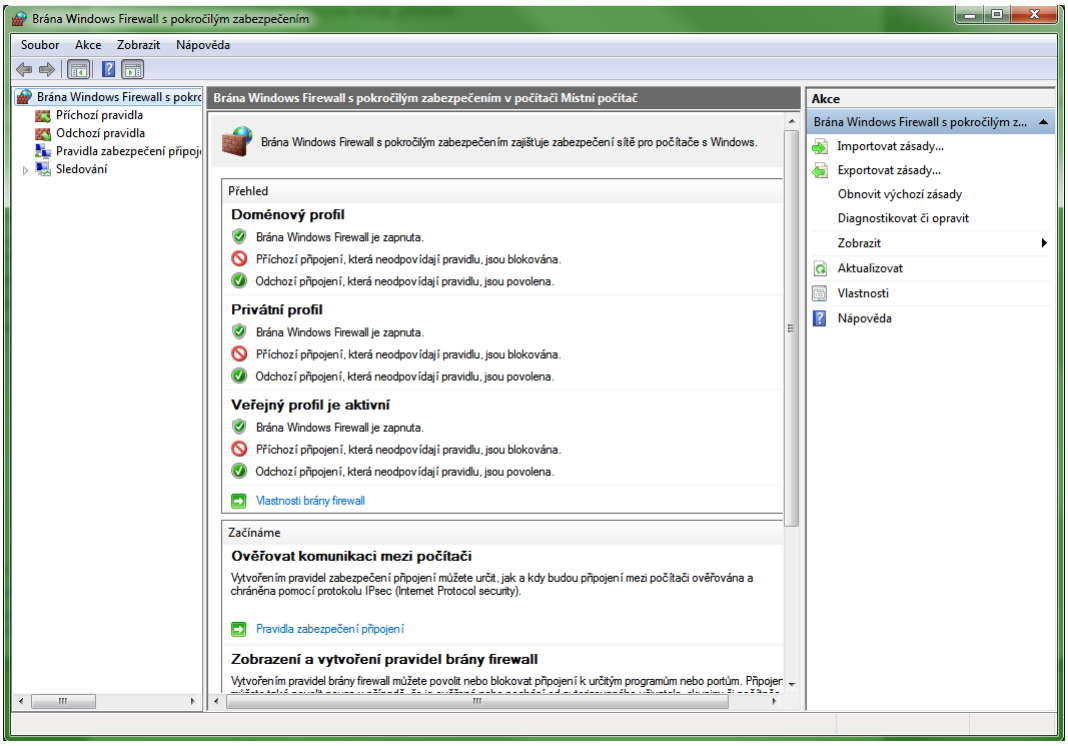
Obr. P 10: Stránka pro nastavení serveru.

Posledním nastavitelným parametrem je *Private addsesses*. Pokud je tento parametr zaškrtnutý, znamená to, že do webového rozhraní se dostanou pouze weboví klienti, kteří jsou na stejné podsíti jako server. Pokud tomu tak nebude, zobrazí se varovná hláška (Connection Error!!! You can not connect to the server from different subnet than server is located.) a klient se nemůže do rozhraní přihlásit.

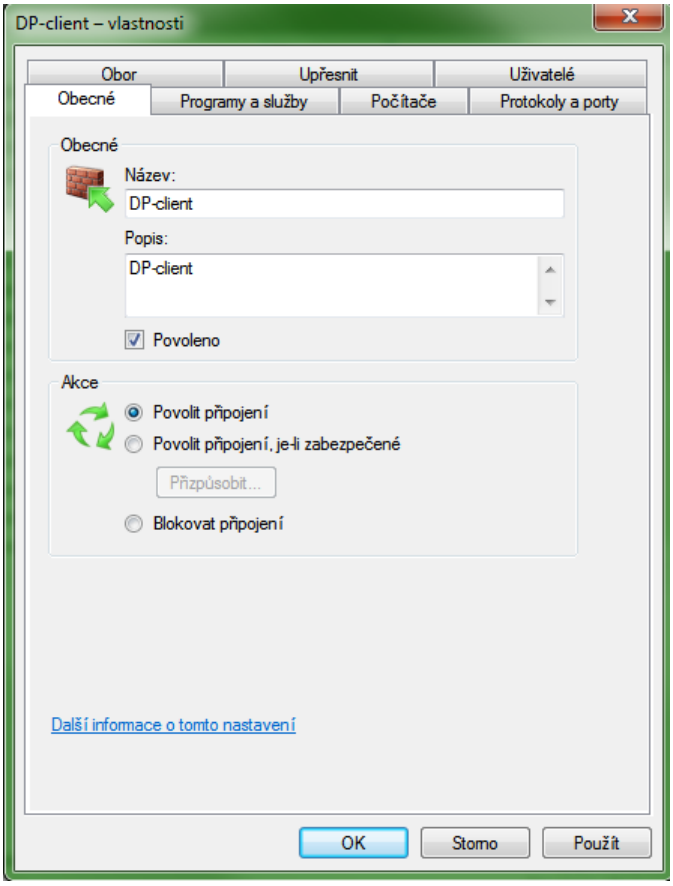
I zde je možné vrátit původní nastavení. To se však nevztahuje na vrácení původního hesla.

Problémy

Při spouštění aplikace se může vyskytnou problém administrátorskými právy. Proto je nunté postupovat přesně podle návodu pro instalaci. Kromě to je možné, že systémový firewall nepropustí příchozí stream. To se dá odstranit tak, že se spustí *Start*, napíše se *firewall* a vybere se odkza *Brána Windows Firewall*. Stejného efektu se dá dosáhnout spuštěním odkazu *Ovládací panely\Systém a zabezpečení\Brána Windows Firewall* v průzkumníku. V otevřeném okně se v levém panelu klikne na *Upřesnit nastavení*. Otevře se nové okno (viz obr. P 11) a zde v levém panelu vybere záložka *Příchozí pravidla*. Zobrazí se seznam aplikací a v něm najděte *DP-client*, klikněte na ní pravým tlačítkem a vyberte vlastnosti. V otevřeném okně, v záložce *Obecné*, překlikněte z *Blokovat připojení* na *Povolit připojení* (viz obr. P 12).



Obr. P 11: Nastavení povolení aplikace ve firewallu systému.



Obr. P 12: Povolení příchozího spojení na klienta.

Konfigurační soubory

Oba programy mají konfigurační soubory, které je možné měnit. Řádky začínající # jsou komentáře a programy je při načítání souboru ignorují.

Server

Server má veškeré nastavení přístupné přes webové rozhraní. Pokud by se do něj z nějakého důvodu nedalo dostat, je ještě možné hodnoty upravit přímo v konfiguračních souborech. Tyto soubory se nachází ve složce *IP_baby_monitor\server\src\server\web_interface\setting*. Upravovat se smějí pouze soubory bez přípony *_o*, tedy *avoption, avset, server*. Soubory s touto příponou obsahují výchozí nastavení a nesmí se ani upravovat ani mazat.

- `avoption` - Obsahuje nastavení limitů pro detekce změn (viz ukázka 6.2). *noise* je hladina zvuku a *frame* je změna obrazu.
- `avset` - Obsahuje nastavení parametrů streamu (viz ukázka 6.3). *resol* je rozlišení obrazu, *bitra* je bit rate, *chann* je počet audio kanálů, *frekv* je frekvence audia, *frame* je počet rámců za sekundu a *quali* je kvalita videa.
- `server` - Obsahuje nastavení serveru (viz ukázka 6.4). *mport* je port, kde server přijímá klienty (přehrávač), *sport* je port pro příjem klientů přes HTTPS, *avport* je port udp serveru, kam je posílán audio+video stream, *aport* je to samé jako avport, ale jen pro audio, *private* false znamená, že se do webového rozhraní mohou přihlásit i klienti z jiné podsítě a *name* je jméno serveru, pod kterým ho lze na podsíti najít.

```
noise=40
frame=10
```

(6.2)

```
resol=800x600
bitra=282000
chann=1
frekv=16000
frame=30
quali=9
```

(6.3)

```
mport=9001
sport=9443
avport=9002
```

```
aport=9004
private=false
name=ip.baby.monitor
```

(6.4)

Klient

Klient má k dispozici jeden konfigurační soubor, který se nedá nastavit jinak, než otevřením v nějakém poznámkovém bloku. Jeho obsah je zobrazen v ukázce 6.5. *SERVER_NAME* je název serveru, *DEFAULT_SERVER_IP* je IP adresa serveru, *SERVER_UDP_AV_SEND_PORT* je port, na kterém server vysílá audio+video stream, *STREAM_SAVE_PORT* je port, kam klient přeposílá stream, pokud se má ukládat do souboru, *MESSAGE_SHOW_DURATION* ke počet sekund, jak dlouho se bude zobrazovat zpráva v oznamovací oblasti, *WINDOW_SHOW_DURATION* je počet sekund, jak dlouho se bude zobrazovat okno se streamem, pokud se objevuje jen při změnách[1].

```
# Server name.
SERVER_NAME=ip.baby.monitor
# Server default IP address.
DEFAULT_SERVER_IP=127.0.0.1
# Udp port where audio/video stream is sent.
SERVER_UDP_AV_SEND_PORT=9002
# Udp port where audio stream is sent.
SERVER_UDP_AUDIO_SEND_PORT=9004
# Udp port where stream is sent and save to file.
STREAM_SAVE_PORT=9006
# How long the tray message will bo shown [s].
MESSAGE_SHOW_DURATION=10
# How long the stream player will play. [s].
WINDOW_SHOW_DURATION=25
```

(6.5)

[1]Tato doba je včetně 10 sekund, po které trvá, než se okno otevře. Pokud se tedy nastaví doba na méně než 11 sekund, okno se pravděpodobně vůbec nestihne otevřít.

Certifikát

Certifikát, který server používá pro vytváření spojení pomocí HTTPS, není ověřen žádnou certifikační autoritou, a tak se při každém přihlášení, bude webový prohlížeč ptát, jestli má pokračovat i přes nedůvěryhodnost certifikátu. Toto se dá odstranit tím, že se daný certifikát, který je ve složce *IP_baby_monitor\certifikat\ip_baby_monitor_cert.pfx*, nainstaluje do počítače mezi *Důvěryhodné kořenové certifikační autority*, a také se přidá do prohlížeče, se kterým se bude do rozhraní přistupovat. Při instalaci bude vyžadováno heslo, které je nastaveno na *1a8G.h6*.

Pro prohlížeč Opera je možné certifikát přidat takto: *Nástroje − > Nastavení... − > Pokročilé volby − > Zabezpečení − > Správce certifikátů − > Importovat*.

Pro prohlížeč Mozilla Firefox je možné certifikát přidat takto: *Nástroje − > Možnosti − > Rozšíření − > Certifikáty − > Certifikáty − > Osobní − > Importovat*.

Pro prohlížeč Chrome je možné certifikát přidat takto: *Nástroje − > Rozšíření − > Nastavení − > Spravovat certifikáty − > Importovat...* .

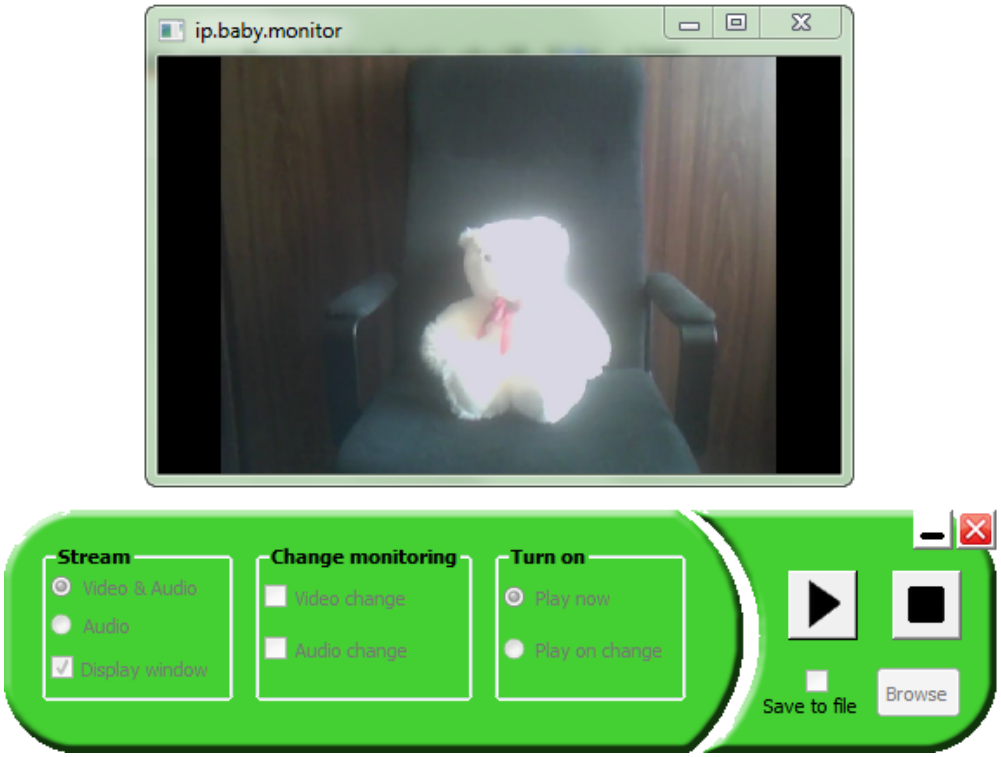
Pro prohlížeč Internet Explorer je možné certifikát přidat takto: *Nastavení − > Možnosti internetu − > Obsah − > Certifikáty − > Importovat...* .

Zobrazovací okno pro stream

Spuštěný stream je přehráván v samostatném okně, které je nezávislé na přehrávači, který je na obr. P 4. To znamená, že pokud se toto okno vypne, přehrávač se to nedozví, a je nutné ho samostatně ukončit tlačítkem *Stop*. Přehrávač i se spuštěným oknem pro stream je na obr. P 13. Pokud se ale ukončí přehrávač, pak by se mělo ukončit i toto okno. Jelikož se jedná o samostatný program, má v sobě implementováno vlastní ovládání. Na toto okno je tedy možné uplatnit následující klávesy:

- q, ESC - vypnutí okna
- f - přepnutí na celou obrazovku

- p, mezerník - pauza
- s - procházení videa rámec po rámci, předtím se musí zmáčknout pauza
- šipka doprava - posunutí na aktuální pozici v live streamu
- w - přepnutí náhledu na zvukové vlny



Obr. P 13: Náhled na spuštěný přehrávač i s oknem se streamem.

Obsah CD

Na přiloženém CD jsou ve složce *IP_baby_monitor* adresáře:

- certifikat - Obsahuje instalační certifikát *ip_baby_monitor_cert.pfx*.
- dokumentace - Obsahuje text k diplomové práci ve formátu pdf.
- klient - Obsahuje veškeré součásti klientského programu jako přeložené soubory, instalační soubor, zdrojové soubory a adresář se spustitelným programem.
- opswi - Obsahuje oborový projekt, který se nad rámec zadání diplomové práce zabývá implementací serveru na zařízení BeagleBone Black s procesorem ARM a přehráváním streamu v XBMC.
- scripty pro instalaci - Obsahuje scripty (Inno Setup), které generují instalační .exe soubory.
- server - Obsahuje veškeré součásti programu serveru jako přeložené soubory, instalační soubor, zdrojové soubory a adresář se spustitelným programem.

Server i klient obsahují stejné podadresáře:

- bin - Obsahuje přeložené soubory všech součástí daného programu (pod Windows).
- install - Obsahuje instalační program.
- run - Obsahuje spustitelnou verzi programu. (To samé je pak ve složce, kam se program nainstaluje.)
- src - Obsahuje zdrojové kódy všech součástí spolu s *MAKEFILE* a knihovnami.

Aby bylo možné přeložit jak server, tak klienta, musí být na systému Windows nainstalována knihovna pro práci s Posix vlákny (viz [22]). Dále je pro oba systémy nutná instalce knihovny *OpenSSL* a pro klienta pak ještě *Qt*. Zdrojové kódy ve složce *server/src/server* se překládájí pomocí *MAKEFILE* pro daný operační systém (*Makefile.Linux* nebo *Makefile.Windows*). Zdrojový kód klienta umístěný v *klient/src/klient* se překládá pomocí *Windows_run.bat* a nebo *Makefile.Linux*.

FFmpeg

Na CD jsou k dispozici i zdrojové soubory pro nástroje *ffmpeg.exe* a *ffplay.exe*. Jejich přeložení pod systémem Windows je realizovatelné a je popsané na fóru, které FFmpeg spravuje [1].