

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2018

Bc. Jaroslav Šalko



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE WEBRTC V OPEN SOURCE PBX

WEBRTC IMPLEMENTATION IN OPEN-SOURCE PBX'S

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Jaroslav Šalko

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Pavel Šilhavý, Ph.D.

BRNO 2018



Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Jaroslav Šalko

ID: 164785

Ročník: 2

Akademický rok: 2017/18

NÁZEV TÉMATU:

Implementace WebRTC v Open source PBX

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte možnosti integrace WebRTC v Open Source PBX. Zaměřte se na Open source PBX Asterisk, FreeSwitch a Kamailio v LTS verzích. Podrobně popište potřebné kroky instalace a konfigurace WebRTC pro uvedené Open source PBX a ve vlastních testech ověřte možnosti a odlišnosti implementací WebRTC v uvedených Open source projektech. Věnujte se rovněž dalším službám, jako např. video a videokonferenční hovory. Vytvořte laboratorní úlohu věnovanou seznámení s problematikou WebRTC v PBX Asterisk.

DOPORUČENÁ LITERATURA:

[1] Meggelen, J.V, Smith, J., Madsen, L. Asterisk™ The Future of Telephony. Sevastopol: O'Reilly Media, Inc., 2005. ISBN 0-596-00962-3.

[2] Minessale, A., Maruzzelli, G. FreeSWITCH 1.8. Packt Publishing, 2017. ISBN 978-1-78588-913-4.

Termín zadání: 5.2.2018

Termín odevzdání: 21.5.2018

Vedoucí práce: Ing. Pavel Šilhavý, Ph.D.

Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Tato práce se zabývá podporou WebRTC komunikace napříč vybranými Open Source PBX. Jmenovitě to jsou PBX Asterisk, FreeSWITCH a Kamailio. Práce je zejména zaměřena na způsob integrace WebRTC v jednotlivých PBX a obsahuje nezbytné kroky pro konfiguraci WebRTC. Práce také obsahuje analýzu šifrované komunikace, pomocí paketového analyzátoru Wireshark, kterou WebRTC vyžaduje. V teoretické části je čtenář seznámen s pojmem WebRTC a s protokoly, které souvisejí s tímto druhem komunikace. Účelem této části práce je přiblížit čtenáři princip fungování a podpůrné protokoly WebRTC. S tím souvisí i popis základních rozhraní WebRTC aplikací.

Dále zde čtenář nalezne konfigurace vybraných Open Source PBX tak, aby byly schopny zprostředkovat audio a video komunikaci mezi WebRTC klienty. Tato část se dá rozdělit na podkapitoly, kdy se každá z nich věnuje tentýž problematice pro jednu z výše uvedených PBX. Na konci každé kapitoly, kde je krok za krokem nakonfigurovaná PBX, jsou provedeny testovací hovory. Tyto hovory jsou zachyceny paketovým analyzátozem Wireshark a slouží jako demonstrace funkčnosti WebRTC konfigurace.

Praktická část obsahuje laboratorní úlohu pro studenty předmětu *telekomunikační a informační systémy*. V této úloze budou studenti krok za krokem konfigurovat WebRTC pro pobočkovou ústřednu Asterisk. Úloha obsahuje stručný popis implementace WebRTC v PBX Asterisk a okomentované kroky, nezbytné pro její konfiguraci. Postup je doplněn o demonstrační ukázky konfiguračních souborů. Dále studenti provedou analýzu šifrované komunikace, což je v případě WebRTC defacto tunelování SIP protokolu v protokolu HTTP.

KLÍČOVÁ SLOVA

WebRTC, PBX, Asterisk, FreeSWITCH, Kamailio, Sipml5

ABSTRACT

The topic of this work is verification of support WebRTC communication through selected Open Source PBX. This work examine demands for WebRTC communications and describes configuration of branch centers for this type of communication. In the theoretical part is reader acquainted with the term WebRTC and with protocols related to this kind of communications. The purpose of this part of the work is to bring the reader closer look to the principles of functioning to ensuring support for this kind of communications. This is also connected with Description of basic interfaces of WebRTC applications.

Further the reader finds the configuration of the selected Open Source PBX so that they can make audio-video call between WebRTC clients. This section is divided into three subchapters, each of it deals with the same problems for one of the aforementioned PBX. At the end of each chapter where the PBX PBX is configured step-by-step, test calls are made. These calls are captured by the Wireshark packet analyzer and serve as a demonstration of the WebRTC configuration functionality. At the end of this section, PBXs are compared against each other about WebRTC support.

Practical part is dealing with laboratory task for students which are studying subject telecommunication and information systems. In the task students will be configuring WebRTC for PBX Asterisk. The task contains brief description of WebRTC and comments for all steps for configuration. All steps and facts are demonstrated by exemplary configuration files.

KEYWORDS:

WebRTC, PBX, Asterisk, FreeSWITCH, Kamailio, Sipml5

ŠALKO, J. *Implementace WebRTC v Open source PBX*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. 98 s. Vedoucí diplomové práce Ing. Pavel Šilhavý, Ph.D..

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Implementace WebRTC v Open source PBX“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno

.....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Pavlu Šilhavému, Ph.D., za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno

.....

podpis autora

Výzkum popsáný v této diplomové práci byl realizován v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

Obsah

Úvod	13
1 Projekt WebRTC	14
2 Základní WebRTC API rozhraní.....	15
2.1 API rozhraní RTCPeerConnection	15
2.1.1 WebSocket	16
2.1.2 SIP over WebSocket	17
2.1.3 XMPP/Jingle	17
2.1.4 XHR/Comet.....	17
2.1.5 Data Channel	17
2.2 API rozhraní MediaStream (getUserMedia)	17
2.3 RTCDataChannel	18
3 WebRTC – Použité mechanismy a protokoly	19
3.1 UDP (User Datagram Protocol)	19
3.2 SDP (Session Description Protocol).....	19
3.3 SCTP (Stream Control Transmission Protocol)	20
3.4 SRTP (Secure Real-time Transport Protocol)	20
3.5 DTLS (Datagram Transport Layer Security)	20
3.6 NAT (Network Address Translation).....	21
3.6.1 Server STUN (Session Traversal Utilities for NAT)	21
3.7 ICE (Interactive Connectivity Establishment)	22
4 Princip komunikace mezi WebRTC klienty.....	22
5 Pobočková ústředna Asterisk	27
5.1 Asterisk a podpora WebRTC	28
5.2 Asterisk, WebRTC dokumentace.....	29
6 FreeSWITCH.....	30
6.1 FreeSWITCH a podpora WebRTC	30
6.2 FreeSWITCH, WebRTC dokumentace	31
7 BPX Kamilio.....	32
7.1 Kamilio a podpora WebRTC.....	32

7.2	Kamailio, WebRTC dokumentace	33
8	Konfigurace pobočkových ústředí	33
8.1	Schéma zapojení	34
8.2	Signalizační protokol.....	34
8.3	WebRTC klient	35
9	Apache server, integrace WebRTC klienta	35
9.1	Implementace Sipml5 klienta na Apache server	35
10	Konfigurace WebRTC v PBX Asterisk	36
10.1	základní predispozice	37
10.2	Konfigurace vestavěného HTTP serveru.....	38
10.3	Konfigurace uživatelských účtů.....	39
10.4	Konfigurace základního kontextu	40
10.5	Příprava prohlížeče.....	41
10.5.1	Konverze certifikátu	41
10.5.2	Import certifikátu do prohlížeče (Mozilla Firefox)	42
10.5.3	Povolení výjimky pro wss transport s naším serverem	42
10.6	Testovací hovor.....	43
10.7	Zachycení a analýza komunikace v programu Wireshark	45
11	Konfigurace WebRTC v PBX FreeSwitch	50
11.1	Konfigurace modulu Sofia, pro naslouchání WebSocket spojení	51
11.2	Konfigurace uživatelských účtů.....	51
11.3	Konfigurace základního kontextu	52
11.4	Příprava prohlížeče.....	53
11.5	Testovací hovor.....	53
11.6	Zachycení a analýza komunikace v programu Wireshark	56
12	Laboratorní úloha - zadání.....	57
12.1	Laboratorní úloha – praktická část	58
	Závěr	59
	Seznam použitých zdrojů	61
	Seznam příloh.....	63
	Příloha A	64

A.1 Instalace Apache serveru.....	64
A.2 Vygenerování vlastního SSL certifikátu.....	65
A.3 Konfigurace Apache pro použití vlastně podepsaného certifikátu	67
Příloha B	70
B.1 Instalace pobočkové ústředny Asterisk 13.....	70
B.2 Vytvoření certifikátu pro TLS.....	72
Příloha C	74
C.1 Instalace pobočkové ústředny FreeSwitch 1.6	74
C.2 Instalace pobočkové ústředny FreeSWITCH 1.6	74
C.3 Vytvoření certifikátu pro TLS.....	76
Příloha D.....	78
D.1 Laboratorní úloha 6 - PBX Asterisk – WebRTC.....	78
Příloha E.....	94
E.1 CD/DVD.....	94
Seznam použitých zkratk.....	95
Seznam tabulek.....	97
Seznam obrázků.....	98

Úvod

Komunikační služby napříč internetovou sítí nejsou žádnou novinkou. Takřka všichni uživatelé počítačů, smartphonů, případně jiných zařízení s přístupem na internet, jistě využívají některou ze služeb pro on-line komunikaci. Služeb, které Vám umožní on-line komunikaci mezi přáteli či kolegy na síti existuje nespočet. Zmínit mohu například ICQ, Jitsi, Google Talk, Facebook Messenger, Viber a mnoho dalších. Tyto služby zpočátku sloužily pouze pro posílání textových zpráv. Již řadu let však on-line komunikace nekončí pouze u textové formy. Tyto služby nabízí hlasovou komunikaci nebo v posledních letech také stále rozmáhající se video hovory nebo dokonce videokonference. Doposud platilo, že pokud jste chtěli využívat potenciál těchto komunikačních služeb naplno, potřebovali jste k tomu specializovaný klientský software. A pokud snad tyto služby disponují svojí online verzí, dostupnou přímo z prohlížeče, v základu nabízejí opět pouze textovou komunikaci. Pro přenos hlasu zachytávaného z mikrofону nebo přenos obrazu, snímaného z webové kamery, je nutné rozšíření v podobě plug-inů. Ať už konkrétním pro danou službu nebo pomocí známého pluginu flash player. Což z pohledu dnešních, stále zvyšujících se nároků na bezpečnost, není zdaleka ideální situace. Tyto plug-iny nezdědka trpí bezpečnostními nedostatky. Mohou způsobovat problémy se stabilitou nebo mohou být dokonce nekompatibilní s tou či onou verzí internetového prohlížeče nebo operačního systému. Společnost Google si byla vědoma tohoto nedostatku a založila projekt WebRTC, jehož cílem je tvůrcům webových aplikací poskytnout takové nástroje, aby byli schopni vytvořit webové aplikace, které budou moci přistupovat k mikrofónu či kameře, bez závislosti na jakémkoliv plug-inu či jiném rozšíření, které není součástí samotného prohlížeče.

Webové aplikace by díky WebRTC mohly nahradit desktopové komunikátory. Což by přineslo výhodu, v podobě plnohodnotného přístupu ke komunikační službě z libovolného počítače s webovým prohlížečem a přístupem na internet, bez nutnosti cokoliv instalovat. Dá se předpokládat, že na základě výše popsaných vlastností WebRTC, budou komunikátory založené na této technologii stále přibývat.

Z tohoto důvodu jsem si vybral právě tuto práci, neboť jejím tématem je prověření, možnosti integrace WebRTC ve vybraných Open Source PBX. Jmenovitě jsou to PBX Asterisk, FreeSWITCH a Kamailio.

1 Projekt WebRTC

WebRTC neboli celým názvem „*Web Real-Time Communications*“ je projekt, který umožňuje multimediální komunikaci v reálném čase v prostředí prohlížečů. Multimediální komunikaci může představovat přenos zvuku, videa nebo dat. WebRTC není první počín, který přináší komunikaci v reálném čase (dále jen RTC) do prostředí prohlížečů. Jako první však umožňuje RTC v rámci prohlížečů bez nutnosti instalace rozšiřujících plug-inů či softwarů třetích stran. Pro implementaci komunikačních technologií přímo do webových prohlížečů, využívá WebRTC API (Application Programming Interface) rozhraní, které přistupuje k Javascript kódu a jazyku HTML5.

WebRTC je projekt společnosti Google, která v květnu roku 2011 zveřejnila projekt s otevřenými zdrojovými kódy, který se zabýval implementací RTC do prostředí prohlížečů. Tento projekt se těšil velkému zájmu, který vedl ke standardizaci relevantních protokolů, které pro prohlížeče definuje a za které odpovídá pracovní skupina IETF (Internet Engineering Task Force). Standardizace protokolů pro WebRTC je popsána příslušnými RFC (Request for Comments) dokumenty, které jsou dostupné online na stránkách společnosti IETF. Uvést mohu například RFC 7742, zabývající se video processingem a použitými kodeky nebo RFC 7874 popisující totéž pro audio. API rozhraní pro prohlížeče je definováno organizací W3C (World Wide Web Consortium). Pracovní verze dokumentu, popisující API pro prohlížeče, byla vydána 26. července roku 2011 pod názvem „*WebRTC 1.0: Real-time Communication Between Browsers*“. Tento dokument je od doby jeho vydání průběžně rozšiřován.

Z počátku bylo WebRTC vyvíjeno přednostně pro prohlížeče Google Chrome a Mozilla Firefox. Postupem času se však implementace WebRTC značně rozšířila a dnes je tento projekt multiplatformní. Seznam prohlížečů a platforem, které podporují WebRTC, je uveden v tabulce níže. Tabulka 1 obsahuje kompatibilní prohlížeče a platformy k začátku roku 2018.

[21]

Tabulka 1 Prohlížeče podporující WebRTC

Prohlížeče podporující WebRTC			
Google Chrome	Mozilla Firefox	Opera	
Platformy podporující WebRTC			
Android	iOS	Firefox OS	Chrome OS

Zdroj: vlastní zpracování

WebRTC se stalo poměrně rychle vizí toho, jak by mělo být řešeno RTC v prohlížečích. Jeho účel je poskytnout kvalitní RTC pro internetové prohlížeče, mobilní platformy a zařízení ze světa IoT (Internet of Things). Koncový uživatel nemusí řešit kompatibilitu se softwarem třetích stran, ani instalaci rozšiřujících balíčků. Vše, co potřebuje, je pouze některý z uvedených podporovaných prohlížečů, dle tabulky 1, v aktuální verzi. Samotná komunikace pak z pohledu uživatele může začít pouhým načtením stránky. [10] [21]

Jak jsem již zmínil, WebRTC je projekt, který spojuje dohromady poměrně rozsáhlý Javascript kód, ke kterému se přistupuje prostřednictvím API rozhraní. To ale není všechno, co WebRTC zahrnuje. Je to také sada standardizovaných protokolů, které řeší lokalizaci peerů (účastníků spojení), vyjednávání parametrů spojení, zabezpečení a tak dále.

2 Základní WebRTC API rozhraní

Jak již bylo uvedeno, WebRTC je sada API rozhraní nad Javascript kódem a jazykem HTML5. WebRTC implementuje tři základní API rozhraní.

1. RTCPeerConnection
2. MediaStream (getUserMedia)
3. RTCDataChannel

2.1 API rozhraní RTCPeerConnection

Navzdory mnoha síťovým protokolům, které zajišťují sestavení, udržování, řízení a ukončení peer-to-peer (dále jen P2P) spojení, z pohledu webové aplikace se jedná o jedno API rozhraní. Rozhraní RTCPeerConnection umožňuje aplikaci vytvořit P2P komunikaci s jinou instancí RTCPeerConnection. Představuje tedy spojení mezi místním a vzdáleným

účastníkem spojení. `RTCPeerConnection` zprostředkovává nastavení připojení, umožňuje správu připojení a poskytuje informace o stavu připojení. Všechny tyto funkce jsou skryty pod rozhraním `RTCPeerConnection`. Souhrnně tedy slouží pro sestavení, udržení a sledování připojení mezi oběma účastníky a ukončí jej, jakmile již není potřeba. Tato komunikace je koordinována pomocí řídicích zpráv, které jsou zprostředkovány blíže nespecifikovaným signalizačním protokolem. WebRTC neimplementuje signalizační protokol z několika důvodů. Různé aplikace mohou využívat různé protokoly a WebRTC má být především univerzálním řešením. WebRTC běží ve webovém prohlížeči a k samotné aplikaci přistupujeme načtením konkrétní stránky. Pakliže by signalizace měla být také řešena ve webovém prohlížeči, vyžadovala by, aby webové stránky byly statické. Signalizace by se jinak totiž rozpadla vždy, při každém opětovném načtení stránky. Z těchto důvodů projekt WebRTC nedefinuje signalizační protokol. Existují však určitá doporučení, která jsou již komunitou prověřená. Jmenovitě se jedná o tato řešení: [5] [10]

1. WebSocket
2. SIP over WebSocket
3. XMPP/Jingle
4. XHR/Comet
5. Data Channel

2.1.1 WebSocket

WebSocket je komunikační řešení, které slouží pro vytvoření obousměrného komunikačního kanálu mezi klientem (aplikace v prohlížeči) a serverem, za pomoci protokolu HTTP. Toto spojení, které využívá transportní protokol TCP, je jednoznačně definováno dvojicí IP adres, použitým portem a protokolem pro komunikaci. Prostřednictvím WebSocketu si mohou účastníci spojení vyměňovat informace v reálném čase, a to dokonce ve full-duplex režimu. Díky spojení prostřednictvím WebSocketu nemá komunikace potíže s průchodem skrze firewall nebo NAT. V případě, kdy během sestavování spojení dochází ke změně IP adres v hlavičkách paketů, se automaticky vytváří mezi koncovými klienty HTTP tunel. Na straně klienta je WebSocket implementován pomocí Javascriptu (tak jako i WebRTC) třídou `WebSocket`. Pro WebSocket URL se používá prefix „ws://“, případně „wss://“, pokud se jedná o komunikaci s využitím SSL (Secure Sockets Layer). Při použití této transportní metody je třeba explicitně definovat, jaké zprávy budeme používat. [5]

2.1.2 SIP over WebSocket

WebRTC popisuje způsob, kdy se prohlížeč stává koncovým bodem komunikace, ale nikoliv jako koncový bod pro SIP. Existují implementace SIP napsané v JavaScriptu, které používají WebSocket pro vytvoření WebRTC relace tak, aby aplikace byla schopna komunikovat se standardními SIP klienty. Tato metoda funguje totožně jako klasický WebSocket, pouze místo původních zpráv přenáší skrze WebSocket standardní SIP komunikaci. V tomto případě můžeme říci, že WebSocket využíváme jako tunel pro přenos zpráv protokolu SIP. [5]

2.1.3 XMPP/Jingle

Jedná se o otevřený protokol pro instant messengery programy, které zprostředkovávají online komunikaci. XMPP slouží pro posílání krátkých zpráv, sestavení privátního hovoru či zobrazení stavu účastníků spojení. Svou funkcionalitou by se mohl přirovnat k protokolu SIP. Navíc oproti protokolu SIP nabízí tzv. transporty, díky nimž je uživatel schopen komunikovat napříč různými moderními komunikačními službami jako např. Skype, Google Talk, LiveJournal Talk a další. [5]

2.1.4 XHR/Comet

Jedná se o model webové komunikace, který umožňuje webovým aplikacím komunikaci mezi serverem a klientem prostřednictvím protokolu HTTP. Při této komunikaci se využívá cíleně dlouho udržovaného spojení za pomoci zprávy „*http request*“, která umožňuje serveru posílat data do webových prohlížečů. [5]

2.1.5 Data Channel

Data channel představuje vlastní komunikační kanál přímo od tvůrců WebRTC. Tento kanál vzniká při prvním spojení mezi dvěma koncovými body komunikace. Jakmile je jednou sestaven, může být využit i pro přenos signalizačních zpráv. Výhodou je, že signalizace skrze data channel nemusí procházet přes žádné signalizační servery. Při použití této metody signalizace je třeba explicitně definovat zprávy, které budeme používat. [5]

2.2 API rozhraní MediaStream (getUserMedia)

Pro získání audio a video dat je třeba zajistit, aby prohlížeč mohl přistupovat k systémovému hardwaru. „*Surová*“ data, neboli data ve formátu RAW však nestačí. Získaná

data je třeba zpracovat za účelem zvýšení kvality, synchronizace a přizpůsobení měnící se datové propustnosti a zpoždění mezi klienty. Všechno toto zpracování probíhá přímo v prohlížeči. Jakmile je tato práce dokončena, webová aplikace obdrží již optimalizovaný mediální stream. API rozhraní `MediaStream`, představuje synchronizovaný proud mediálního toku dat. Tento proud obsahuje tzv. stopy (`tracks`), které reprezentují audio nebo video data. Každá stopa je definovaná jako instance `MediaStreamTrack`. Objekt `MediaStream` se můžeme skládat z nula či více objektů `MediaStreamTrack`. `MediaStreamTrack` může obsahovat jeden nebo více mediálních toků. Je to stejné jako například zvukový kanál přidružený pravému a levému reproduktoru. Objekt `MediaStream` má vždy pouze jeden vstup a jeden výstup. Objekt `MediaStream`, získaný pomocí volání metody `MediaDevices.getUserMedia()`, se nazývá lokální. Tato metoda poskytuje přístup ke vstupním zařízením, které jsou zdrojem dat (`MediaStreamTrack`) pro `MediaStream`. Vstupním zařízením může být kamera nebo mikrofón. Objekt `MediaStream` může být sestaven i z nelokálních zdrojů. V tomto případě objekt `MediaStream` obsahuje nula či více `MediaStreamTrack`, které nepochází z lokálních zdrojů, ale z API rozhraní `RTCPeerConnection`. [5] [15] [17]

2.3 RTCDatChannel

Rozhraní `RTCDatChannel` představuje obousměrný síťový kanál, pro výměnu aplikačních dat mezi oběma účastníky komunikace. `RTCDatChannel` je vytvořen pomocí jedné ze základních metod objektu `RTCPeerConnection`. `RTCDatChannel` je svým fungováním velmi podobný `WebSocketu`. Na rozdíl od `WebSocketu`, který vytváří spojení klient-server, `RTCDatChannel` vytváří P2P spojení. Umožňuje spojení dvou aplikací skrze jediné TCP spojení. `RTCPeerConnection` může teoreticky nabývat až 65 534 síťovými kanály pro `RTCDatChannel`. Rozhraní `RTCDatChannel` spolupracuje s těmito protokoly: [5] [15]

- UDP – poskytuje P2P přenos dat
- DTLS – poskytuje šifrování přenášených dat
- SCTP – poskytuje řízení toku dat

3 WebRTC – Použité mechanismy a protokoly

Komunikace v reálném čase je značně citlivá na zpoždění. Oproti tomu krátkodobé přerušení datového toku má minimální dopad na kvalitu výstupu. RTC služby proto kladou důraz především na nízké zpoždění než na spolehlivost. Požadavek na nízké zpoždění je hlavním důvodem, proč je protokol UDP preferovaným transportním protokolem pro doručování dat v reálném čase. Jinak tomu není ani v případě WebRTC, který rovněž pro přenos audio a video dat používá transportní protokol UDP. V případě WebRTC ale nemůžeme data pouze zapouzdřit do UDP datagramu a poslat na síť. WebRTC také potřebuje mechanismy pro překonání mnoha vrstev překladačů NAT a firewallů po trase, vyjednání parametrů spojení, poskytnutí šifrování dat, řízení toku a další. Pro splnění všech požadavků WebRTC, potřebuje prohlížeč podporovat také velkou škálu podpůrných protokolů a služeb. [7] [5] [10] [21]

3.1 UDP (User Datagram Protocol)

Jedná se o protokol transportní vrstvy referenčního modelu ISO/OSI. Protokol UDP je označován za nespolehlivý způsob doručení zpráv k cíli. Protokol UDP nezaručuje úspěšné doručení zprávy, nezasílá potvrzení o přijetí, neprovádí opakované vysílání, nezavádí číslování datagramů, nesleduje stav připojení. Naproti tomu je díky své jednoduchosti vhodný pro nasazení aplikací pracujících systémem dotaz-odpověď nebo tam, kde se počítá se ztrátami datagramů a není vhodné, aby se ztrácel čas opětovným odesíláním.

3.2 SDP (Session Description Protocol)

SDP je protokol, který slouží k vyjednání budované relace multimediálního přenosu dat. Tento protokol nepřenáší samotná data, slouží pouze pro vyjednání parametrů, které se při následném přenosu použijí. Parametrem může být například IP adresa, port, podporované datové formáty (tj. audio, video kodeky), transportní protokol (RTP, UDP, atd.).

3.3 SCTP (Stream Control Transmission Protocol)

Tento protokol se tak jako UDP či TCP nachází na transportní vrstvě. Původně byl tento protokol navržen pro přenos signalizačních zpráv SS7 v IP sítích. Tento protokol se od ostatních transportních protokolů liší zejména navázáním spojením. Při navázání spojení, které v terminologii SCTP nese název asociace, vzniká několik navzájem nezávislých kanálů. V rámci jednotlivých kanálů dokáže protokol SCTP garantovat doručení všech dat cíli ve správném pořadí. Případná ztráta některých dat, a jejich následné znovu-odeslání, nemá vliv na přenos dat v ostatních kanálech. SCTP tak jako TCP zajišťuje spolehlivý přenos dat.

3.4 SRTP (Secure Real-time Transport Protocol)

I dnes zdaleka ne všechna VoIP zařízení podporují šifrování. Proto, aby se zamezilo odposlouchávání RTP komunikace po trase mezi dvěma účastníky, byl vyvinut protokol SRTP. SRTP je protokol, který poskytuje šifrování RTP streamu a přidává možnost ověřovat zprávy a integritu RTP relace.

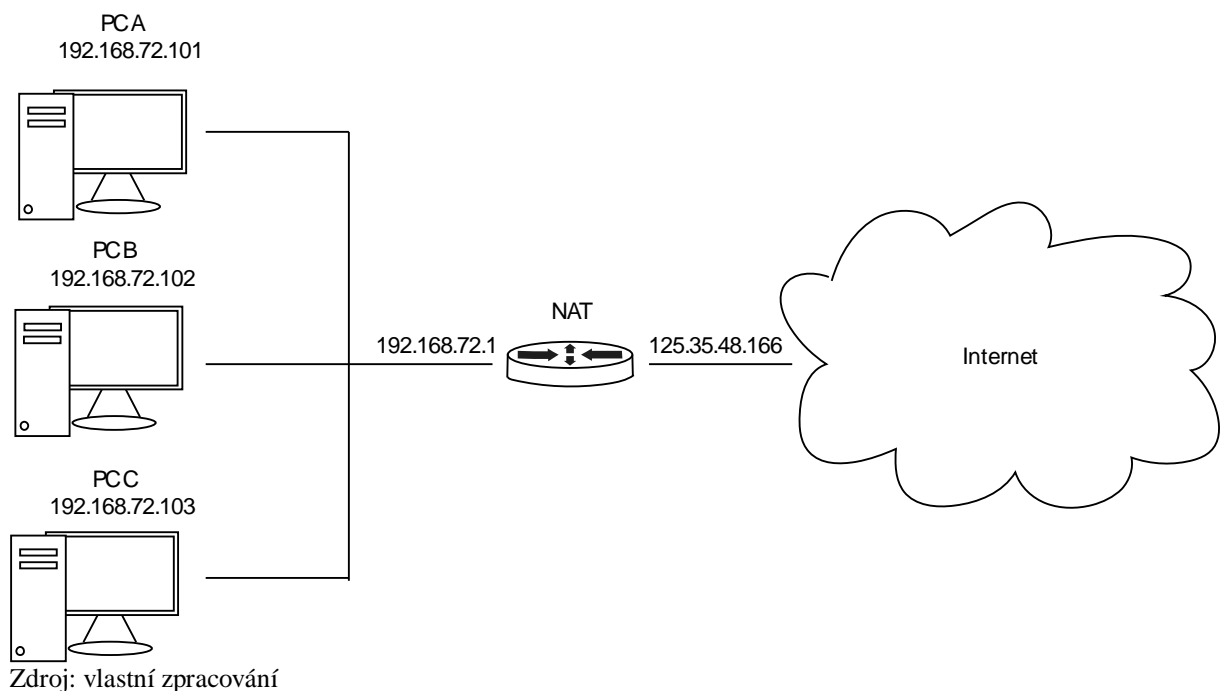
3.5 DTLS (Datagram Transport Layer Security)

Tento protokol se v informatice používá pro zabezpečení datagramových protokolů, jako je například transportní protokol UDP. Protokol DTLS umožňuje aplikacím s transportním protokolem UDP komunikovat způsobem, který je určen k zabránění odposlouchávání, manipulace nebo padělání zpráv. Protokol DTLS je založen na protokolu TLS (Transport Layer Security) a poskytuje totožné záruky. Důvod, proč nemůžeme použít přímo protokol TLS je ten, že TLS nemá žádné vnitřní mechanismy pro znovunavázání spojení. V případě ztráty některých dat je v případě TLS spojení rozpojeno a označeno za nespolehlivé. Tato vlastnost je zcela nepřijatelná pro fungování v datagramovém prostředí. DTLS pouze patřičně modifikuje TLS tak, aby byl vyřešen tento problém.

3.6 NAT (Network Address Translation)

NAT je metoda zajišťující přístup stanic z lokální sítě, s neveřejnou IP adresou, na veřejnou síť (internet). NAT upravuje hlavičky protokolů síťové vrstvy (přepisem zdrojové nebo cílové IP adresy), případně upravuje i hlavičky protokolů vyšších vrstev (přepisem čísla portů). Díky těmto úpravám umožňuje NAT vystupovat více počítačům s různou lokální IP adresou na internetu pod jednu veřejnou IP adresou. Níže, na **Obrázek 1**, vidíme tři počítače v lokální síti 192.168.72.0/24. Každý z PC má svojí unikátní lokální IP adresu. Pokud však chtějí komunikovat mimo svojí lokální síť, posílají požadavek na svojí výchozí bránu 192.168.72.1, což je router s funkcí NAT, který přepisuje jejich lokální IP adresy na veřejnou adresu 125.35.48.166. Uvedený příklad je označován jako Full-cone NAT (Masquerade). Existuje však více druhů překladu síťových adres (Address-restricted-cone, Port-restricted-cone, symmetric NAT).

Obrázek 1 NAT - demonstrační ilustrace

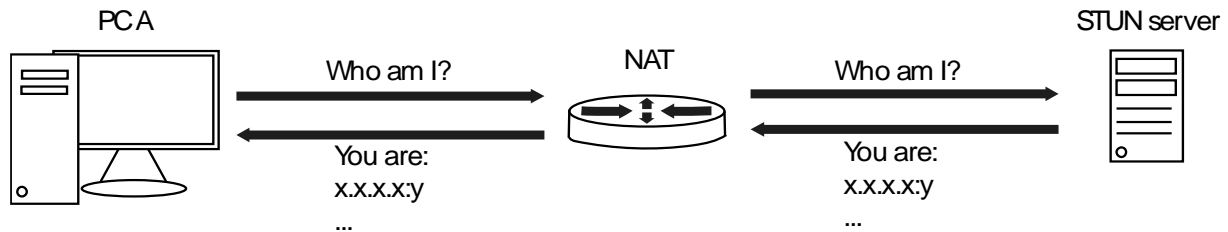


3.6.1 Server STUN (Session Traversal Utilities for NAT)

Server STUN umožňuje klientům zjistit jejich veřejnou IP adresu, typ překladu NAT a veřejný („internetový“) port, přiřazený k překladači NAT s určitým lokálním portem.

Tyto informace se používají k sestavení hovoru mezi klientem a poskytovatelem VoIP služby. STUN tedy poskytuje důležité informace pro komunikaci skrze překladače síťových adres (NAT). Tím pádem umožňuje sestavovat telefonní hovory s poskytovatelem VoIP služby, který se může nacházet i mimo lokální síť. Nutno zmínit, že server STUN není použitelný pro všechny typy překladu NAT. STUN pracuje pouze s použitím překladu NAT typu full-cone nebo address/port restricted-cone.

Obrázek 2 STUN - demonstrační ilustrace



Zdroj: vlastní zpracování

3.7 ICE (Interactive Connectivity Establishment)

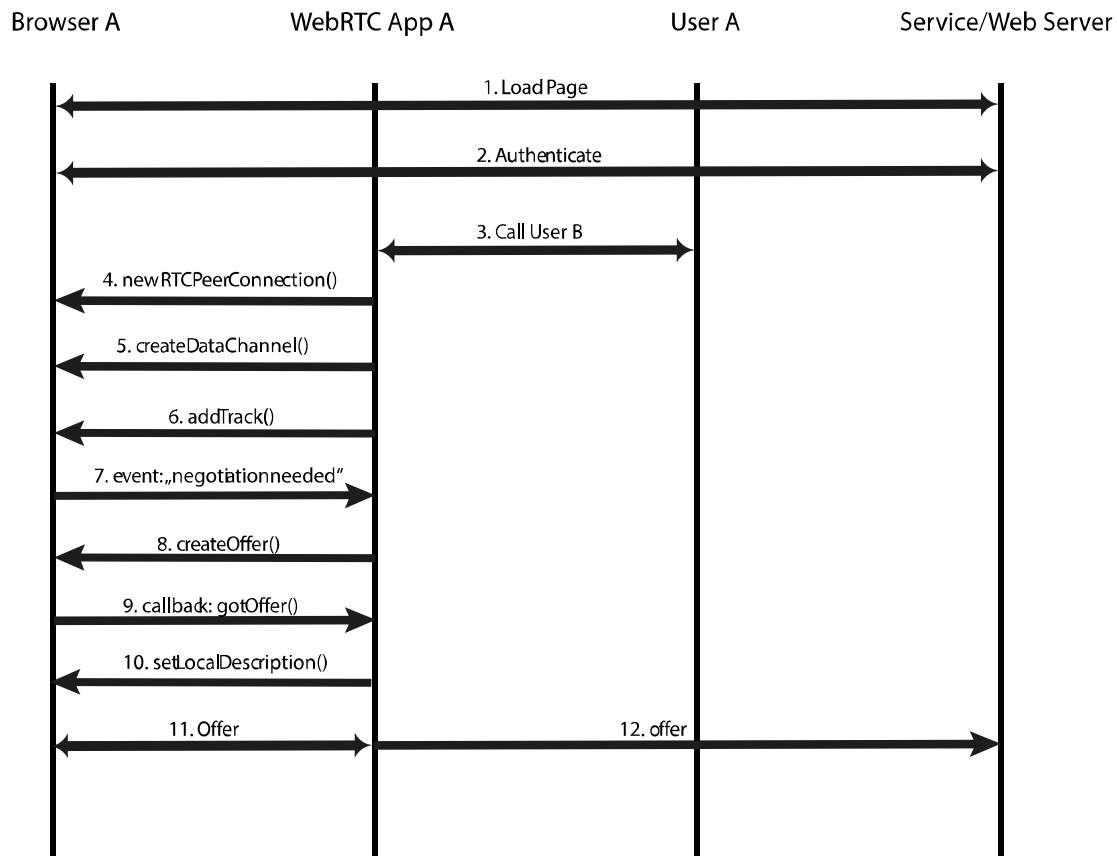
VoIP, P2P spojení a mnoho dalších aplikací vyžadují informace o cílových účastnících spojení, a nevystačí si pouze s adresami v hlavičkách paketů IP protokolu. ICE je technika, která se využívá v počítačových sítích k nalezení nejvhodnějších STUN a TURN serverů, pro překonání síťových překladů NAT. Tyto informace slouží pro sestavení P2P komunikace mezi dvěma účastníky spojení. Tato technika s využitím STUN, popř. TURN serverů, umožňuje WebRTC klientům překonat složitost reálných sítí. P2P spojení nevyužívá ke zprostředkování komunikace žádný centrální server, proto je velice obtížné toto spojení sestavit skrze internet, kde je třeba projít přes překladače NAT a Firewally.

4 Princip komunikace mezi WebRTC klienty

Na **Obrázek 3** si můžete prohlédnout, jak vzniká počáteční sestavení komunikace. WebRTC používá API `RTCPeerConnection` pro vytvoření spojení mezi prohlížeči. Toto API rozhraní má při inicializaci spojení, co se signalizace týče, několik úkolů. Jedním z nich je zjištění komunikačních schopností WebRTC klientů, jako jsou například podporované kodeky, formáty, schopnosti rozlišení atd. Tyto informace se použijí při sestavení zprávy typu „offer“, resp. „answer“ a slouží k popisu SDP relace. Toto vše se odehrává v rámci

prohlížeče. Po zjištění těchto vlastností je nutno tyto informace předat protistraně. [5] [23]

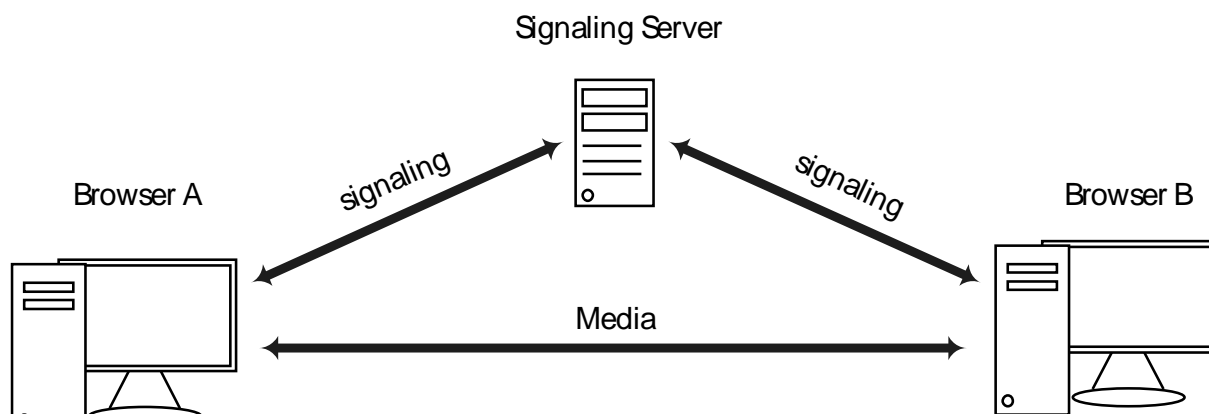
Obrázek 3 WebRTC - schéma navazování spojení 1



Zdroj: vlastní zpracování

V praxi nejsme schopni sestavit P2P spojení, aniž bychom předem protistranu kontaktovali. I kdybychom znali přesné umístění protistrany v síti, znali IP adresu a port, stejně nejsme schopni ověřit a zajistit, zda je protistrana připravena s námi komunikovat. Proto se ani WebRTC neobejde bez signalizačních serverů, které slouží pro navázání počáteční komunikace. V ideálním případě, jak demonstruje **Obrázek 4**, by každý WebRTC klient měl svojí vlastní unikátní IP adresu a port, přes které by bylo možné komunikovat s ostatními uživateli. I v tomto případě je však zapotřebí signalizačního serveru, skrze který bude moci WebRTC klient říci ostatním, na které adrese jej mohou kontaktovat.

Obrázek 4 WebRTC - signalizace v ideálních podmínkách

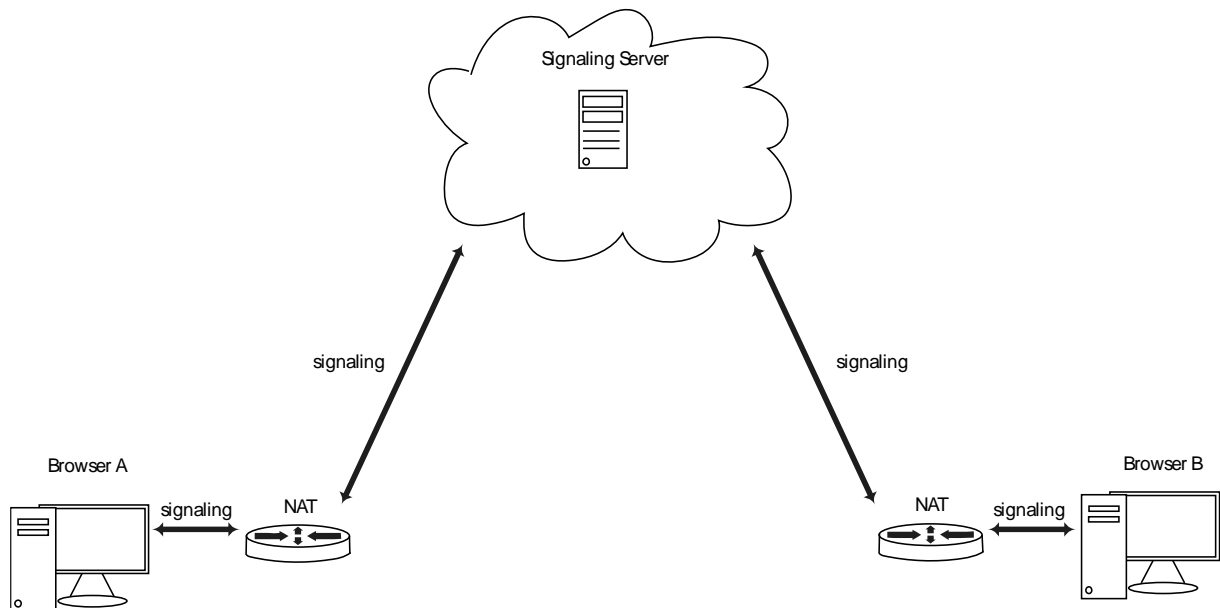


Zdroj: vlastní zpracování

WebRTC klient A kontaktuje signalizační server s požadavkem na spojení s WebRTC klientem B (zpráva „offer“ z **Obrázek 3**). Signalizační server ověří existenci uživatele v registračním záznamu uživatelů na registračním serveru, kde jsou registrováni všichni WebRTC klienti využívající danou službu. Klient B na základě přeposlané zprávy „offer“, stejným způsobem jako klient A vytvořil zprávu „offer“ (viz předchozí **Obrázek 3**), vytvoří zprávu „answer“ a přepoše jí skrze signalizační server klientovi A. Jakmile se skrze signalizaci domluví oba klienti na společných parametrech, které použijí pro sestavení P2P spojení, dojde k navázání spojení mezi API `RTCPeerConnection` obou klientů a může začít samotný přenos mediálních dat.

V reálných podmínkách, jak demonstruje **Obrázek 5**, je však většina zařízení maskována za jednu nebo více vrstvami síťového překladače NAT. Také se mohou nacházet za proxy servery nebo firewally, které blokují určité porty a protokoly. Všechny tyto prvky mohou být implementovány v jednom zařízení, jako například v domácím wifi routeru. Poté není tak jednoduché navázat P2P spojení pouze za využití signalizačního serveru.

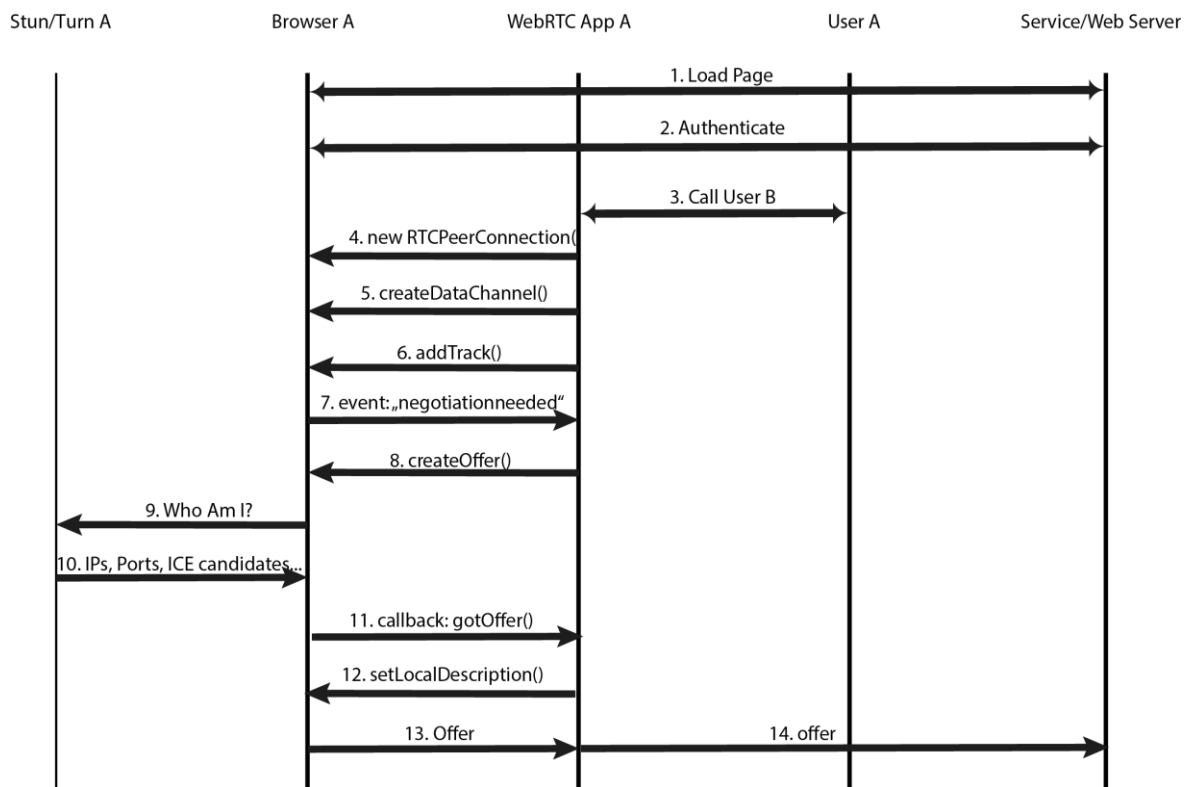
Obrázek 5 WebRTC - signalizace v reálných podmínkách



Zdroj: vlastní zpracování

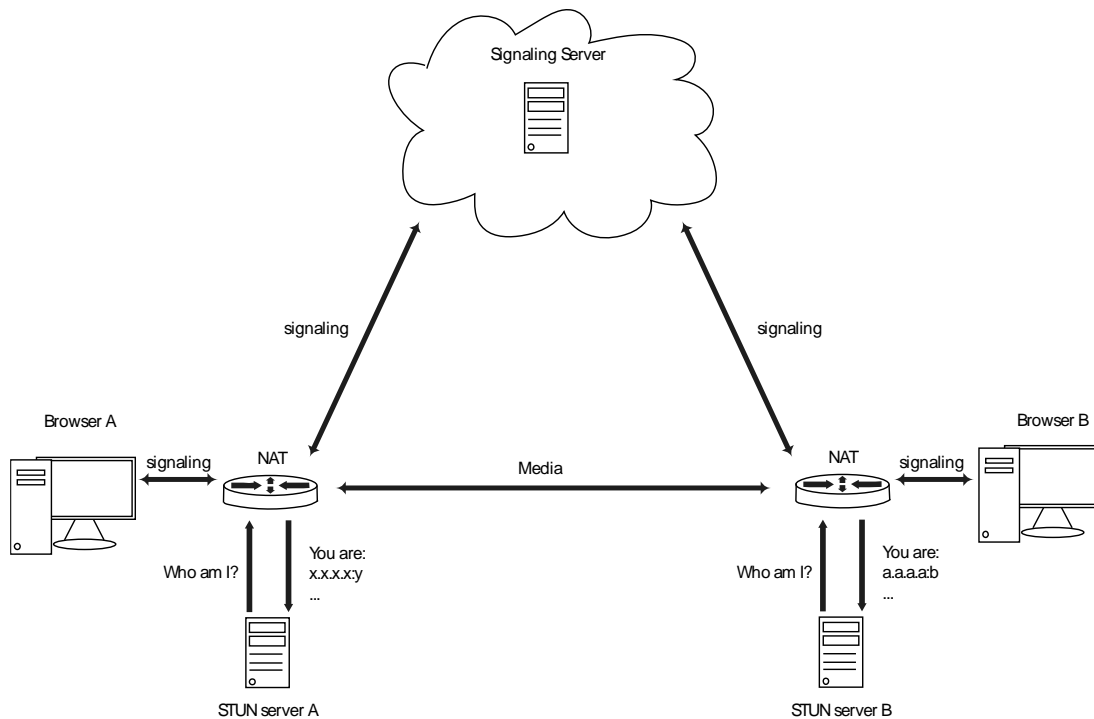
V případě uvedeném na **Obrázek 6**, API rozhraní `RTCPeerConnection` klienta A, dle výsledků analýzy typu NAT mapování realizovanou metodou ICE, kontaktuje STUN server, který mu poskytne potřebné informace pro jeho identifikaci ve veřejné síti. Ve většině případů je STUN server používán pouze během sestavování spojení. Po navázání této relace je médium sestaveno přímo mezi klienty.

Obrázek 6 WebRTC - schéma navazování spojení 2



Zdroj: vlastní zpracování

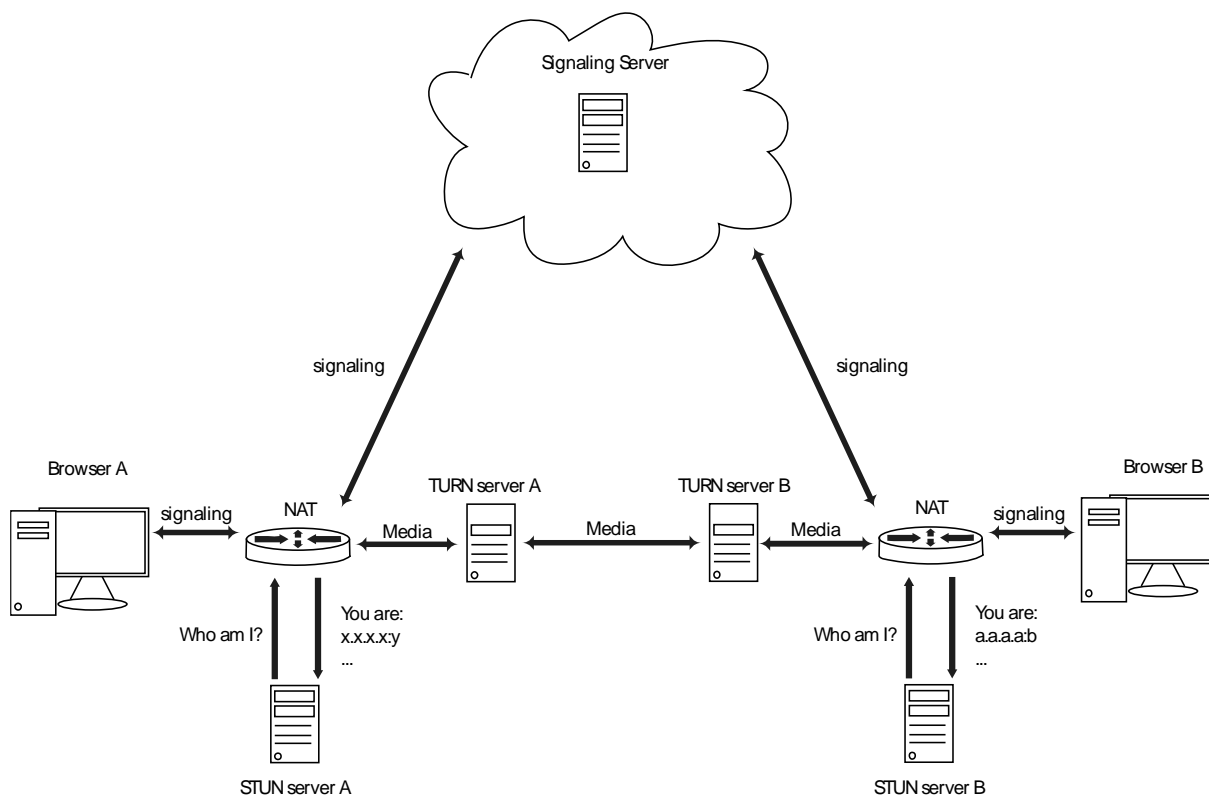
Obrázek 7 WebRTC - signalizace v reálných podmínkách (STUN)



Zdroj: vlastní zpracování

Pokud se STUN serveru nepodaří zjistit veřejnou IP adresu, port a typ překladu NAT klienta A, obrací se ICE na TURN server. TURN je rozšířením STUN serveru, které lépe umožňuje průchod médií skrze překladače NAT. Na rozdíl od STUN serveru, TURN server zůstává v cestě mezi klienty i po sestavení spojení. TURN server tedy v komunikaci zastává funkci media proxy serveru, jak je znázorněno na **Obrázek 8**.

Obrázek 8 WebRTC - signalizace v reálných podmínkách (TURN)



Zdroj: vlastní zpracování

5 Pobočková ústředna Asterisk

Asterisk je Open-Source software, který umožňuje implementaci telefonní ústředny pomocí běžného HW. Původně byl Asterisk vyvíjen pouze jako telefonní systém pro malou firmu. Dnes se jedná o univerzální „nástroj“ pro budování telefonních systémů. V dnešní době Asterisk nenalezneme pouze u IP PBX, ale i ve VoIP bránách, systémech call center, konferenčních řešeních, hlasové poště a mnoha dalších aplikacích, které zahrnují komunikaci v reálném čase. Toho je docíleno díky více než desetiletému působení Asterisku na trhu.

Asterisk má kolem sebe širokou komunitu, která dle informací na oficiálních stránkách projektu, čítá přes 86 000 registrovaných uživatelů, vývojářů a advokátů, kteří přispěli k tomu, že dnes Asterisk patří mezi jeden z nejrozsáhlejších komunikačních projektů na světě.

Asterisk je napsán v jazyce C, což z něj dělá modulární a konfigurovatelný nástroj. Architektura se skládá z jádra, které může komunikovat s mnoha moduly. Tyto moduly poskytují jádru ovladače, díky kterým je program Asterisk schopen dle předdefinovaných pravidel do jisté míry řídit chování externích programů či zařízení tak, aby mezi nimi usnadnil komunikaci. Kromě funkcí poskytovaných samotným jádrem Asterisku, mohou rozšiřující moduly nabízet uživatelům další funkce. Většina modulů je distribuována přímo od vývojářů Asterisku, ačkoli jiné moduly mohou být dostupné od členů komunity nebo firem, které vytvářejí své vlastní komerční moduly. Moduly jsou volitelné a při instalaci či načítání Asterisku můžete ovlivnit, které moduly chcete použít a které ne. [1][4]

5.1 Asterisk a podpora WebRTC

Asterisk podporuje WebRTC od verze 11, která byla vydána v druhé polovině roku 2012. Implementace WebRTC je zajištěna pomocí integrovaného HTTP serveru, který slouží pro zpracování HTTP požadavků a nových modulů, umožňující komunikaci přes WebSocket. PBX Asterisk používá pro WebRTC transportní metodou SIP over WebSocket. Z počátku Asterisk podporoval pro WebRTC pouze hlas. Od verze 14 přibývají i kodeky pro podporu videa. Samotní tvůrci však WebRTC video stále označují za projekt ve vývoji. Tomu napovídá i fakt, že za LTS verzi je stále označena verze 13. Asterisk ve verzi 15 přidává podporu videokonferencí typu SFU (Selective Forwarding Unit). To znamená, že Asterisk bude podporovat videokonference s vybranými komunikátory, které SFU implementují, např. mediasoup, kurento, Janus, Jitsi a další. Dle vývojářů následující verze Asterisku, Asterisk 16, bude LTS verzí. [1] [2] [3]

Tabulka 2 PBX Asterisk - podpora WebRTC

Kritéria	Asterisk version 13.X LTS
Podpora WebRTC	ANO
WebRTC signalizace	SIP
Transportní metoda	SIP over WebSocket
Vestavěný HTTP server	ANO
Podpůrné protokoly pro WebRTC	ICE, STUN, TURN, SRTP, AVPF
WebRTC audio	ANO
Podporované audio kodeky pro WebRTC	g711, g722
WebRTC video	ANO
Podporované video kodeky pro WebRTC	VP8
WebRTC videokonference	NE
JavaScript knihovny	JsSIP, sipml5

Zdroj: vlastní zpracování

Pro potřeby podpory WebRTC musí být Asterisk sestaven s těmito moduly:

- `res_crypto` – poskytuje možnosti použití kryptografického podpisu, s využitím knihovny OpenSSL.
- `res_http_websocket` – implementuje vestavěný HTTP server, který zajišťuje podporu pro komunikaci skrze WebSocket.
- `res_pjsip_transport_websocket` – definuje jakousi pomyslnou obálku pro `pjsip_transport`. Díky tomuto modulu mohou entity `pjsip endpoints` komunikovat skrze WebSocket.

5.2 Asterisk, WebRTC dokumentace

Na oficiálních stránkách Asterisku je přímo od vývojářů uveden návod, jak nakonfigurovat PBX Asterisk s podporou WebRTC. [24]

Tento návod je dostatečně stručný a obsahuje všechny nezbytné informace. Konfigurace neobsahuje žádné pokročilejší funkce a jedná se opravdu o základní nastavení. Dále je tu stále aktivním fórum, kde komunita i samotní vývojáři sdílí své znalosti. Je zde k nalezení spousta

vláken zabývající se problematikou okolo WebRTC. I zde můžou případní zájemci nalézt odpovědi na své otázky, případně požádat o pomoc. [4]

6 FreeSWITCH

FreeSWITCH je dalším ze zástupců open-source PBX. Tato ústředna vychází z původní PBX Asterisk, ovšem přináší v určitých směrech jistá vylepšení. Projekt FreeSWITCH vznikl v roce 2006 a stojí za ním skupina vývojářů PBX Asterisk, kteří nesouhlasili s politikou a směrem, kterým se původního projekt Asterisk dále ubíral. Dnes se na tomto projektu podílí mnoho dalších vývojářů a uživatelů. Projekt byl od počátku koncipován se zaměřením na modularitu, škálovatelnost a jednoduchost podpory mezi různými platformami. Díky tomu FreeSWITCH nabízí bohatou podporu nejrůznějších kodeků, pokročilých funkcí a možnost propojení s různými komunikačními technologiemi jako např. Skype, H.323 nebo WebRTC.

FreeSWITCH se skládá ze stabilního jádra, které je napsáno v programovacím jazyce C a nad kterým je postaveno API rozhraní. Na toto jádro se dále vážou vzájemně nezávislé moduly. Tyto moduly ve většině případů nepotřebují znát, jaké další moduly jsou právě aktivní nebo jak se chovají. Pomocí těchto modulů lze funkce, které FreeSWITCH poskytuje, dále rozšiřovat. Moduly komunikují s jádrem skrze zprávy „events“ nebo skrze API volání. Obecně systém fungování PBX FreeSWITCH je koncipován tak, aby zvládal větší zatížení. [9]

6.1 FreeSWITCH a podpora WebRTC

FreeSWITCH podporuje WebRTC od verze 1.4 beta, která byla vydána na počátku roku 2014. Tato verze přinesla podporu SIP over WebSocket, nezbytnou pro WebRTC komunikaci, do základního modulu mod_sofia. Mod_sofia zajišťuje koncovým bodům SIP komunikace interakci s jádrem ústředny. Pro zpracování HTTP požadavků slouží modul mod_httapi a mod_http_cache. Ve verzi 1.6 byl také implementován vlastní signální protokol Verto, který přináší zjednodušené kódování a implementaci hovorů, mezi verto komunikátory a koncovými body na straně PBX FreeSWITCH. Verto je tedy alternativou k SIP v Javascriptu, přenášenému skrze WebSocket, jak tomu u WebRTC komunikace

nejčastěji bývá. FreeSWITCH současně podporuje obě alternativy WebRTC komunikace. Jak standardní SIP komunikaci, tak i vlastní Verto. Skrze Verto jsou však podporovány jak WebRTC videohovory, tak i videokonference, sdílení obrazovek nebo přijímání a odesílání dat v reálném čase. [9] [22]

Tabulka 3 PBX FreeSWITCH - podpora WebRTC

Kritéria	FreeSWITCH version 1.6.X LTS
Podpora WebRTC	ANO
WebRTC signalizace	SIP, Verto
Transportní metoda	SIP/Verto over WebSocket
Vestavěný HTTP server	NE
Podpůrné protokoly pro WebRTC	ICE, STUN, TURN, SRTP, AVPF
WebRTC audio	ANO
Podporované audio kodeky	g711, g722, iSAC, iLBC
WebRTC video	ANO
Podporované video kodeky	VP8
WebRTC videokonference	ANO
JavaScript knihovny	JsSIP, sipml5, verto

Zdroj: vlastní zpracování

Pro potřeby podpory WebRTC musí být FreeSWITCH sestaven s těmito moduly:

- mod_sofia – jedná se o základní modul PBX FreeSWITCH, který byl rozšířen o podporu ws a wss transportu. Tento modul umožňuje koncovým bodům SIP komunikace interakci s jádrem ústředny.
- mod_verto – jedná se o modul poskytující alternativní signalizační protokol pro WebRTC.

6.2 FreeSWITCH, WebRTC dokumentace

Na oficiálních stránkách FreeSWITCH je přímo od vývojářů uveden návod, jak nakonfigurovat PBX FreeSWITCH s podporou WebRTC. [22]

Tento návod obsahuje nezbytné informace pro konfiguraci. Návod je stručný a je zaměřen zejména na použití s veřejně podepsanými certifikáty. FreeSWITCH již při instalaci poskytuje základní vestavěné certifikáty pro WebSocket, které lze v krajních

případech použít. Odkaz na konfiguraci slouží zároveň i jakési fórum, kde mohou uživatelé vkládat své dotazy přímo pod článek/návod. Konfigurace neobsahuje žádné pokročilejší funkce a jedná se opravdu o základní nastavení.

7 BPX Kamailio

Kamailio je Open Source projekt, který lze využít pro realizaci velkých systémů pro VoIP a komunikaci v reálném čase. Počátky Kamailia sahají do roku 2001, kdy projekt ještě nesl název OpenSER (SIP Express Router). Cílem tvůrců bylo vytvořit jeden z nejvýkonnějších SIP proxy serverů, který bude zvládat tisíce hovorů za sekundu. Výkon Kamailia umožňuje efektivně řešit problémy s provozní zátěží, jako mohou být poruchy síťových komponentů, útoky nebo rychle rostoucí počet požadavků na server, například po restartu. Kamailio se proto často používá jako SIP proxy pro rozšíření SIP-to-PSTN bran, PBX systémů nebo mediálních serverů, jako jsou Asterisk nebo FreeSWITCH. Tento software může realizovat SIP server, SIP proxy server, SIP registrar server, SIP location server, SIP application server, SIP Websocket server nebo SIP redirect server. [16]

7.1 Kamailio a podpora WebRTC

Kamailio, podobně jako PBX Asterisk, může obsahovat vestavěný HTTP server pomocí rozšiřujícího modulu xHTTP. Tento modul slouží pro zpracování HTTP požadavků. Další z nezbytných modulu pro podporu WebRTC je modul WEBSOCKET, který umožňuje využívat WebSocket jako transportní metodu. Podpora obou modulů přišla s verzí 4.0.0. Nicméně Kamailio podporuje WebRTC pouze ze signalizačního hlediska, kdy nijak nezpracovává mediální tok. Samotné nastavení jak HTTP serveru, tak WebSocketu se provádí v hlavním konfiguračním souboru kamailio.cfg. [7] [16]

7.2 Kamailio, WebRTC dokumentace

Na oficiálních stránkách Kamailia, přímý návod na konfiguraci serveru pro podporu WebRTC není. Mají však vedené dokumentace ke všem modulům, jako např. k xHTTP a WebSocket, které jsou pro WebRTC nezbytné. V mnoha případech dokumentace obsahuje i ukázky konfigurací. Konfigurace samotných modulů je ponechána zcela na uživateli. Konfigurace se navíc od ostatních ústředen liší tím, že se zde nepracuje s definovanými parametry, které je třeba povolovat nebo nastavovat. Uživatel musí sám definovat chování modulu. Pro modul xHTTP vývojáři doporučují vycházet z RFC 6454. Pro WebSocket je to poté RFC 6455, případně ještě zmiňují RFC 7118 a RFC 7977. [7] [16]

Tabulka 4 PBX Kamailio - podpora WebRTC

Kritéria	Kamailio version 5.1.X LSV
Podpora WebRTC	ANO
WebRTC signalizace	SIP
Transportní metoda	SIP over WebSocket
Vestavěný HTTP server	ANO
Podpůrné protokoly pro WebRTC	ICE, STUN, TURN
WebRTC audio	-
Podporované audio kodeky	-
WebRTC video	-
Podporované video kodeky	-
WebRTC videokonference	-
JavaScript knihovny	JsSIP, sipml5

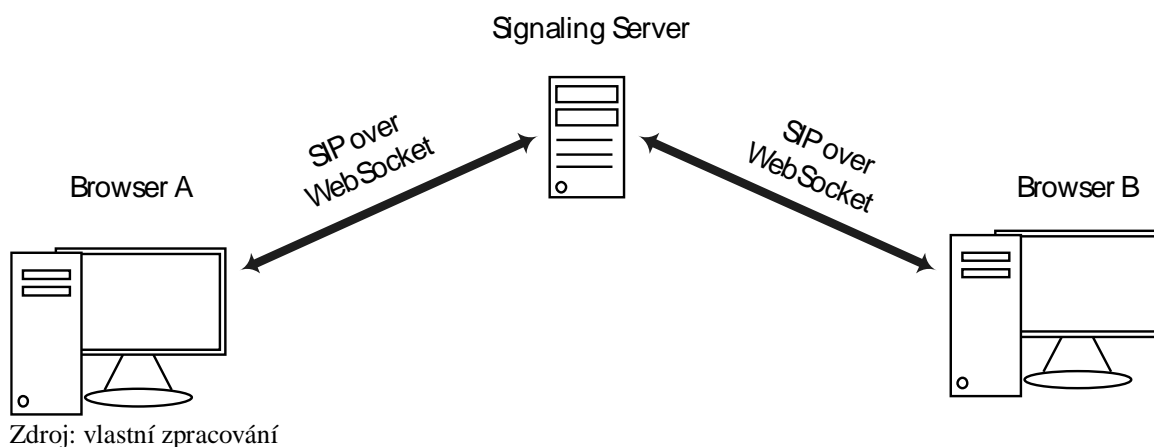
Zdroj: vlastní zpracování

8 Konfigurace pobočkových ústředen

Tato část se zabývá konfigurací WebRTC, již dříve uvedených PBX, na základě dokumentace z jejich oficiálních stránek.

8.1 Schéma zapojení

Obrázek 9 Schéma zapojení



Zapojení se skládá z PC, na kterém je provozován virtuální operační systém Ubuntu, ve verzi 16.04. Ubuntu je hostitelským operačním systémem signalizačního serveru a Apache serveru. Funkce signalizačního serveru budou plnit vybrané PBX, kterým se tato práce věnuje. Apache server bude sloužit klientům pro přístup na WebRTC klienta. V síti se dále nacházejí klienti, kteří jsou zastoupeny smartphony s operačním systémem android, ve verzi 7.1 a vyšší a notebooky s operačním systémem Windows 7 a Windows 10. Tito klienti budou přes rozhraní svých internetových prohlížečů přistupovat na Apache server s WebRTC klientem. Všechna zařízení se nacházejí ve stejné síti, bez přístupu na internet.

8.2 Signalizační protokol

Jak již bylo zmíněno, komunikace skrze WebRTC začíná počáteční signalizací. Existuje více protokolů, které je možno použít pro přenos signalizačních zpráv. Jak je patrné již z **Obrázek 9**, jako signalizační protokol jsem si zvolil standardní SIP. Učinil jsem tak zejména protože s tímto protokolem umí pracovat spousta starších systému poskytující VoIP služby. Také co se týče kompatibility se staršími analogovými zařízeními, můžeme předpokládat,

že všechny brány sloužící pro připojení těchto systémů do VoIP sítě podporují SIP protokol.

Pro transport SIP komunikace mezi ústřednou a WebRTC klientem jsem zvolil transportní metodu WebSocket.

8.3 WebRTC klient

Poslední součástí je použití některého z volně dostupných WebRTC klientů. Na internetu lze jistě dohledat řadu WebRTC klientů. Na základě přehledné dokumentace a široké komunity jsem jako vhodné klienty vybral Sipml5 a JsSIP. Oba klienti na svých oficiálních stránkách mají aktivní fórum, kde lze nalézt věcné příspěvky. Pro účely mé práce jsem v případě těchto dvou klientů zvolil WebRTC klienta Sipml5. Sipml5 byl jeden z prvních WebRTC klientů vůbec. [19]

9 Apache server, integrace WebRTC klienta

V této kapitole je popsána integrace Sipml5 klienta na webový Apache server. Tento klient bude sloužit pro uživatele jako veřejně dostupný komunikátor. Pro potřeby instalace Apache serveru s vlastně podepsanými SSL certifikáty, můžete použít **Příloha A**. Na konci této kapitoly, by měl být čtenář schopen načíst adresu svého Apache serveru s integrovaným WebRTC klientem.

9.1 Implementace Sipml5 klienta na Apache server

V původním nastavení se při načtení IP adresy našeho serveru zobrazuje základní stránka definovaná souborem `index.html` v adresáři `/var/www/html/`.

Proto, abychom implementovali Sipml5 klienta na náš server, přejdeme na webové stránky <http://www.doubango.org/sipml5> a stáhneme si volně dostupné zdrojové soubory toho klienta. [19]

Stažený .zip soubor vyextrahujeme a přejdeme do adresáře `sipml5-master`. V tomto adresáři nalezneme konfigurační soubory. V adresáři odstraníme původní soubor `index.html`. Dále přejmenujeme původní soubor `call.htm` na `index.html`. Nyní všechny soubory přesuneme do adresáře `/var/www/html/` a nahradíme původní `index.html` v tomto adresáři novým.

Po provedení výše popsaných změn, provedeme restart služby Apache, aby se projevil změny.

```
sudo systemctl restart apache2
```

Nyní můžeme otevřít libovolný webový prohlížeč a přejít na adresu apache serveru využívající zabezpečený přístup: `https://<FQDN_or_server_IP>`. Budeme upozorněni, že server, na který přistupujeme, nevyužívá certifikát podepsaný od ověřených certifikačních autorit. Přidáme výjimku pro server a měla by se nám načíst registrační stránka Sipml5 klienta.

Obrázek 10 Sipml5 klient - okno registrace

The screenshot shows the Sipml5 client interface. At the top, there is a dark navigation bar with a 'Home' link and a logo. The main content area is divided into two panels. The left panel, titled 'Registration', contains several input fields: 'Display Name' (with example 'e.g. John Doe'), 'Private Identity' (with example 'e.g. +33600000000'), 'Public Identity' (with example 'e.g. sip:+33600000000@doubango.org'), 'Password', and 'Realm' (with example 'e.g. doubango.org'). Below these fields are 'Login' and 'Logout' buttons. A note indicates that the 'Public Identity' field is mandatory. At the bottom of this panel are two buttons: 'Need SIP account?' and 'Expert mode?'. The right panel, titled 'Call control', has a sub-header 'Video enabled'. It features a text input field labeled 'Enter phone number to call' and two buttons: 'Call' and 'HangUp'. At the bottom left of the interface, there is a copyright notice: '© Doubango Telecom 2012-2016 Inspiring the future'.

Zdroj: vlastní zpracování

10 Konfigurace WebRTC v PBX Asterisk

V této kapitole je popsána konfigurace WebRTC pro pobočkovou ústřednu Asterisk v LTS verzi 13. V případě, že Asterisk server nemáte nainstalovaný, můžete využít **Příloha B**. Pro úspěšnou konfiguraci WebRTC je zapotřebí, aby Asterisk server obsahoval

moduly `res_crypto`, `res_http_websocket` a `res_pjsip_transport_websocket`. Veškerý zde uvedený postup byl proveden na virtualizovaném operačním systému Ubuntu 16.04. Použité příkazy jsou psané takovou formou, aby je bylo možné použít metodou „copy-paste“. Na konci této kapitoly, by měl být čtenář schopen provést za pomoci PBX Asterisk, jakožto signalizačního serveru, úspěšný testovací hovor s využitím WebRTC klienta Sipml5.

10.1 základní predispozice

Aby byl Sipml5 klient schopen komunikace s PBX Asterisk, musí být ústředna nainstalována s podporou těchto kanálových ovladačů:

- `res_crypto` – zajišťuje zdroje pro kryptografické aplikace
- `res_http_websocket` – podpora transportní metody WebSocket pro `chan_sip`
- `res_pjsip_transport_websocket` – podpora transportní metody WebSocket pro `chan_pjsip`

Chcete-li zkontrolovat, zda máte tyto moduly dostupné, můžete použít následující příkaz:

```
# ls -w 1 /usr/lib/asterisk/modules/{*crypto*,*websocket*}
```

Měli byste dostat tento výstup:

```
/usr/lib/asterisk/modules/res_crypto.so
/usr/lib/asterisk/modules/res_http_websocket.so
/usr/lib/asterisk/modules/res_pjsip_transport_websocket.so
```

Dále zkontrolujte, zda má Asterisk tyto moduly načtené:

```
# asterisk -rx "module show like crypto"
# asterisk -rx "module show like websocket"
```

Měli byste dostat tento výstup:

```
# asterisk -rx "module show like crypto"
Module          Description                               Use Count    Status
res_crypto.so   Cryptographic Digital Signatures         1            Running
1 modules loaded

# asterisk -rx "module show like websocket"
Module          Description                               Use Count    Status
res_http_websocket.so  HTTP WebSocket Support                 3            Running
res_pjsip_transport_websocket.so  PJSIP WebSocket Transport Support 0      Running
2 modules loaded
```

Pokud nemáte moduly načteny, zkontrolujte konfigurační soubor `/etc/asterisk/modules.conf` a ujistěte se, že nemáte tyto moduly explicitně vypnuty. Dále je

třeba zajistit, aby modul `chan_sip.so` nenaslouchal WebSocket spojení. Vložte proto do sekce „*general*“ v konfiguračním souboru `/etc/asterisk/sip.conf` tento parametr:

```
websocket_enabled=false
```

10.2 Konfigurace vestavěného HTTP serveru

Asterisk disponuje vestavěným HTTP serverem, který budeme využívat pro otevření portu, na kterém bude proces asterisk naslouchat a navazovat WebSocket spojení. Pro jeho konfiguraci slouží soubor `http.conf` umístěný v adresáři `/etc/asterisk/`. [24]

a) Editujte soubor `http.conf`, dle přiložené konfigurace

```
[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlsbindaddr=0.0.0.0:8089
tlscertfile=/etc/asterisk/keys/asterisk.pem
```

Touto konfigurací jsme povolili procesu asterisk naslouchání na portu 8088, resp. 8089. Zajistili jsme podporu TLS a povolili přístup ze všech ethernetových rozhraní |Asterisk serveru. Jako certifikát, pomoci nějž, se bude komunikace ověřovat, používáme certifikát `asterisk.pem`, umístěný v adresáři `/etc/asterisk/keys/asterisk.pem`.

Po úpravách provedených v konfiguračním souboru `http.conf`, restartujte HTTP modul z konzole Asterisku.

```
reload http
```

Zda proces asterisk naslouchá na příslušném portu, můžeme ověřit zadáním níže uvedeného příkazu v konzoli asterisku.

```
http show status
```

Měli byste dostat tento výstup:

```
HTTP Server Status:
Prefix:
Server: Asterisk/certified/13.13-cert4
Server Enabled and Bound to 0.0.0.0:8088
```

10.3 Konfigurace uživatelských účtů

Pro konfiguraci uživatelských účtů slouží v PBX Asterisk soubor `pjsip.conf`. V tomto souboru se blíže specifikují veškeré parametry jednotlivých uživatelů. Uživatelský účet musí být nakonfigurován tak, aby jako transportní metodu využíval WebSocket. V tomto případě budeme pro WebSocket používat identifikátor přenosu „*Secure WebSocket*“ `wss`. Pro definování transportní metody, kterou budou uživatelé využívat, upravte soubor `/etc/asterisk/pjsip.conf` dle následujícího postupu:

```
[transport_wss]
type=transport
protocol=wss
bind=0.0.0.0
```

Nyní, když už máme definovanou transportní metodu, musíme vytvořit entitu v rámci `pjsip.conf`, která bude představovat koncový uživatelský účet, ke které budeme přistupovat za pomoci Sipml5 klienta z internetového prohlížeče. Tato entita se bude skládat celkem ze tří objektů. `Aor`, `auth` a `endpoint`. Pro konfiguraci této entity v rámci `pjsip.conf`, rozšířte soubor `/etc/asterisk/pjsip.conf` o následující řádky:

```
[1000]
type=aor
max_contacts=1
remove_existing=yes

[1000]
type=auth
auth_type=userpass
username=1000
password=1000
```

Těmito řádky zajistíme, že entita bude známá jako „*1000*“ a rovněž bude používat „*1000*“ pro ověření uživatelského jména a hesla. **UPOZORŇUJI**, že toto není bezpečné heslo pro použití ve veřejné síti.

Dále musíme vytvořit koncový bod/objekt, který bude odkazovat na objekty aor a auth jako na svoje konfigurační parametry. Pro konfiguraci koncového bodu doplníme soubor */etc/asterisk/pjsip.conf* o následující:

```
[1000]
type=endpoint
transport=transport_wss
aors=1000
auth=1000
use_avpf=yes
media_encryption=dtls
dtls_ca_file=/etc/asterisk/keys/ca.crt
dtls_cert_file=/etc/asterisk/keys/asterisk.pem
dtls_verify=fingerprint
dtls_setup=actpass
ice_support=yes
media_use_received_transport=yes
rtcp_mux=yes
context=default
disallow=all
allow=alaw, VP8
```

Kompletní popis všech parametrů je uveden na manuálové stránce projektu Asterisk. [24] Jen v krátkosti uvedu, co jsme právě nastavili:

- vytvořili jsme koncový bod, který se odkazuje na námi vytvořené objekty aor a auth z přechozího kroku;
- `use_avpf=yes` zajišťuje, aby byl použit profil AVPF, který podporuje protokol SRTP;
- jako šifrovací metodu jsme zvolili DTLS, ke které jsou spjaty naše certifikáty a klíče;
- povolili jsme podporu mechanismu ICE;
- `media_use_received_transport=yes` říká Asterisku, aby pro odesílání dat využil stejný transport, skrz který data obdržel;
- `rtcp_mux=yes` povoluje přenos RTP a RTCP eventů skrze stejný socket;
- `context=default` definuje, že příchozí požadavky na tento koncový bod, budou obslouženy dle kontextu „*default*“;
- explicitně povolujeme pouze kodeky alaw a ulaw.

10.4 Konfigurace základního kontextu

Abychom mohli provést zkušební hovor, je třeba vytvořit základní kontext (směrovací plán) s jednoduchým pravidlem, pomocí kterého si ověříme volání prostřednictvím WebRTC.

Níže uvedené řádky definují kontext „*default*“, který při vytočení čísla 200 přehraje jeden ze základních zvukových souborů a poté hovor zavěsí. Dále je zde základní pravidlo pro volání linky 1000 a 2000, při vytočení identických čísel, za pomoci PJSIP stacku. Tuto úpravu proveďte v souboru `/etc/asterisk/extensions.conf`

```
[default]
exten=> 200,1,Answer()
exten=> 200,2,NoOp(Dovolali jste se na ustrednu)
exten=> 200,3,Playback(demo-congrats)
exten=> 200,4,Hangup()

exten =>1000,1,Dial(PJSIP/1000)
exten =>2000,1,Dial(PJSIP/2000)
```

Nyní jsme provedli základní konfiguraci v pobočkové ústředně Asterisk, která by měla umožnit spojení s WebRTC klientem skrze WebSocket. Aby došlo k aplikování všech změn, které jsme provedli ve směrování, je třeba z konzole Asterisku provést restart modulu dialplan:

```
dialplan reload
```

10.5 Příprava prohlížeče

Ve výchozím nastavení prohlížeče Chrome a Firefox není umožněno připojení pomocí Secure WebSocketu k serveru, který používá certifikát s vlastním podpisem. Lepší variantou by bylo provozovat Asterisk na veřejné IP adrese a nainstalovat certifikát podepsaný od veřejné, známe certifikační autority. Tuto možnost v této ukázce nemáme, a tak musíme importovat vlastně podepsaný certifikát do prohlížeče.

10.5.1 Konverze certifikátu

Protože prohlížeče standardně nepodporují certifikáty ve formátu `.pem`, který jsme si vygenerovali pomocí Asterisku, budeme muset konvertovat náš vlastně podepsaný certifikát do některého z podporovaných formátů. Standardní formáty, které jsou podporované internetovými prohlížeči jsou formáty `.pfx` nebo `.p12`. Pro konverzi certifikátu využijeme nástroj OpenSSL. Samotou konverzi provedeme dle následujícího postupu:

a) Přejdeme do adresáře, kde se nachází Asteriskem vygenerované klíče a certifikáty

```
cd /etc/asterisk/keys
```

b) V tomto adresáři použijeme příkaz nástroje OpenSSL pro konverzi mezi formáty

```
openssl pkcs12 -export -out <název_výstupního_certifikátu>.p12 -inkey  
asterisk.key -in asterisk.crt -certfile ca.crt
```

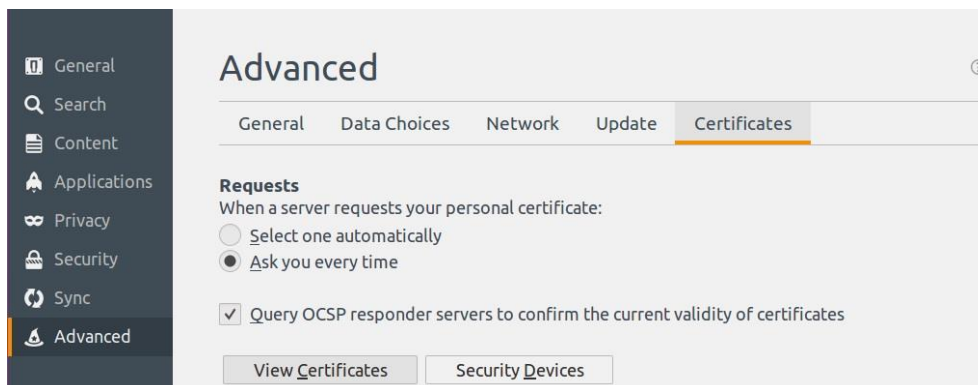
Po zadání příkazu budete vyzváni k zadání hesla, které bude součástí námi vytvořeného certifikátu. Zadejte heslo, které jste nastavili při generování certifikátu. Ve složce `/etc/asterisk/keys` byste měli najít nově vytvořený certifikát `<název_výstupního_certifikátu>.p12`, který jste vytvořili v bodě b.

10.5.2 Import certifikátu do prohlížeče (Mozilla Firefox)

Nový certifikát ve formátu.p12 je třeba naimportovat do prohlížeče. Uvedený postup je aplikovatelný na prohlížeč Mozilla Firefox.

a) V pravém horním rohu otevřeme nastavení (tři vodorovné pruhy) a přejdeme do nastavení certifikátu: *Settings -> Preference -> Advanced -> View Certificates*

Obrázek 11 Mozilla Firefox - import certifikátu 2



Zdroj: vlastní zpracování

V nastavení certifikátů přejdeme do záložky „Your Certificates“, klikneme na tlačítko „Import“ a vyhledáme a naimportujeme náš certifikát ve formátu .p12, v adresáři `/etc/asterisk/keys`

10.5.3 Povolení výjimky pro wss transport s naším serverem

Abychom povolili wss transport pro náš server s vlastně podepsaným certifikátem, je třeba přejít na stránku `https://<IP_Asterisk_serveru>:8089/ws` a zde přidělit tomuto spojení výjimku.

10.6 Testovací hovor

Přejdeme na stránky našeho Apache serveru. WebRTC klienta nakonfigurujeme dle následujícího postupu, který je uveden pro linku 1000. V kolonce „*Registration*“ vyplníme následující údaje:

- Display Name: 1000
- Private Identity*: 1000
- Public Identity*: sip:1000@<IP_Asterisk_serveru>
- Password: 1000
- Realm*: <IP_Asterisk_serveru>

Poté přejdeme do expert módu, kliknutím na tlačítko „*Expert mode?*“ a specifikujeme náš WebSocket pro transport. Kolonku „*WebSocket Server URL*“ vyplňte následujícím způsobem:

- WebSocket Server URL: wss://<IP_Asterisk_serveru>:8089/ws

Nastavení uložíme kliknutím na tlačítko „*Save*“ a vrátíme se na stránku s oknem registrace. Při stisknutí tlačítka „*LogIn*“, byste měli být nahoře informováni o úspěšném zaregistrování statusem „*Connected*“.

Pokud se podíváme do konzole Asterisku, pomocí příkazu, uvidíme něco v tomto smyslu:

```
== WebSocket connection from '192.168.1.218:37324' for protocol 'sip' accepted using version '13'  
-- Added contact 'sips:1000@192.168.1.218:37324;transport=ws;rtcweb-breaker=no' to AOR '1000'  
with expiration of 200 seconds  
== Contact 1000/sips:1000@192.168.1.218:37324;transport=ws;rtcweb-breaker=no has been created  
== Endpoint 1000 is now Reachable
```

Z výpisu vyčteme, že bylo sestaveno nového WebSocket spojení s koncovým bodem 192.168.1.218:37324, které bude použito pro přenos protokolu SIP. Tento koncový bod přistupuje/registruje se jako pjsip endpoint 1000, využívající transportní metodu ws.

- a) Stejným způsobem zaregistrujte z jiného zařízení WebRTC klienta k účtu 2000.
- b) Proveďte testovací video-hovor mezi klienty 1000 a 2000, pomocí ovládacího okna Sipml5 klienta. Otevřeme si nabídku „Call“ a vybereme „Video“.
- c) Budeme dotázáni, zda chceme dané stránce/aplikaci povolit přístup k mikrofonu. Povolení přidělíme kliknutím na tlačítko „Allow“.
- d) Následně byste měli v ovládacím okně Sipml5 klienta vidět v levém horním rohu status „Call in progress“
- e) Při úspěšném sestavení SIP komunikace, bude status odpovídat stavu „In Call“.
V tomto stavu již dochází k RTP přenosu.

V konzoli Asterisku byste měli vidět podrobný průběh hovoru. Ten by měl být podobný níže uvedenému.

```

== Setting global variable 'SIPDOMAIN' to '192.168.1.49'
== DTLS ECDH initialized (automatic), faster PFS enabled
== DTLS ECDH initialized (automatic), faster PFS enabled
  -- Executing [2000@default:1] Answer("PJSIP/1000-00000000", "") in new stack
  -- Executing [2000@default:2] Dial("PJSIP/1000-00000000", "PJSIP/2000") in new stack
== DTLS ECDH initialized (automatic), faster PFS enabled
== DTLS ECDH initialized (automatic), faster PFS enabled
  -- Called PJSIP/2000
  -- PJSIP/1000-00000000 requested media update control 26, passing it to PJSIP/2000-
00000001
  -- PJSIP/2000-00000001 is ringing
  -- PJSIP/2000-00000001 answered PJSIP/1000-00000000
  -- Channel PJSIP/2000-00000001 joined 'simple_bridge' basic-bridge <283fc56a-24ed-4302-
a289-b8c7190c46cb>
  -- Channel PJSIP/1000-00000000 joined 'simple_bridge' basic-bridge <283fc56a-24ed-4302-
a289-b8c7190c46cb>
  -- Channel PJSIP/1000-00000000 left 'simple_bridge' basic-bridge <283fc56a-24ed-4302-a289-
b8c7190c46cb>
== Spawn extension (default, 2000, 3) exited non-zero on 'PJSIP/1000-00000000'
  -- Channel PJSIP/2000-00000001 left 'simple_bridge' basic-bridge <283fc56a-24ed-4302-a289-
b8c7190c46cb>
  -- Removed contact 'sips:1000@192.168.1.218:37766;transport=ws;rtcweb-breaker=yes' from
AOR '1000' due to Request
== Contact 1000/sips:1000@192.168.1.218:37766;transport=ws;rtcweb-breaker=yes has been
deleted
== Endpoint 1000 is now Unreachable
== WebSocket connection from '192.168.1.218:37766' forcefully closed due to fatal write error

```

Z konzole můžeme vyčíst, že automatickým výběrem nejlepší možné šifry, byla použita šifra DTLS ECDH pro výměnu klíčů pro šifrování. Pokračujeme vstupem do číslovacího plánu s názvem „default“ a dotazujeme se na 2000. Číslovací plán, postupuje krok za krokem

dle předepsaných pravidel, které jsme definovali v `extensions.conf`. V tomto případě je to vyhledání příslušného směrovacího pravidla, v kontextu „*default*“, při dotázání na 2000 a následně volání PJSIP endpoint 2000, za pomoci PJSIP stacku.

10.7 Zachycení a analýza komunikace v programu Wireshark

Pro analýzu WebRTC komunikace budeme potřebovat Wireshark ve verzi 2.6.0 a vyšší. Neboť Wireshark od této verze podporuje analýzu dešifrovaných HTTP request sekvencí. Pokud budeme chtít analyzovat WebRTC komunikaci, v našem případě analyzovat SIP zprávy, které přenášíme na stranu WebRTC klienta přes WebSocket, budeme muset mírně upravit náš HTTP server na straně Asterisku. Standardně se při výměně klíčů (tzv. TCP handshake) používá „*nejsilnější*“ společná šifra, mezi klientem a serverem. Mezi takové patří zejména šifry využívající algoritmus ECDH (Elliptic-curve Diffie–Hellman). Šifry využívající ECDH algoritmy znemožňují ověřit autentičnost klienta, a tedy i přes fakt, že vlastníme soukromý klíč v nezašifrované podobě, se nám nepodaří komunikaci dešifrovat. Proto je nutné vynutit striktně použití šifry serveru, kterou předem omezíme na takovou, která nevyužívá ECDH algoritmus.

a) Nakonfigurujeme HTTP server dle přiložené konfigurace

```
[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlscipher=AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA
tlsservercipherorder=yes
tlsbindaddr=0.0.0.0:8089
tlscertfile=/etc/asterisk/keys/asterisk.pem
```

Pokud si zaznamenáme síťový provoz mezi Asterisk serverem a WebRTC klientem v čase registrace a uskutečnění testovacího hovoru, uvidíte při otevření nahrávky něco podobného, jako na níže uvedených obrázcích. Na **Obrázek 12** vidíme pouze zachycenou TCP komunikaci. Informace o SIP protokolu jsou zašifrované v těle TCP segmentů (pole „*Data*“), jak je zobrazeno na **Obrázek 13**.

Obrázek 12 Wireshark - šifrovaná SIP komunikace

1524	67.645338	192.168.1.218	192.168.1.49	TCP
1526	67.646816	192.168.1.49	192.168.1.218	TCP
1532	67.853127	192.168.1.218	192.168.1.49	TCP
1537	67.895749	192.168.1.218	192.168.1.49	TCP
1539	67.898099	192.168.1.49	192.168.1.218	TCP
1544	67.905982	192.168.1.49	192.168.1.218	TCP
1556	68.125239	192.168.1.218	192.168.1.49	TCP
1569	68.428795	192.168.1.49	192.168.1.72	TCP
1577	68.553083	192.168.1.72	192.168.1.49	TCP
1578	68.592629	192.168.1.72	192.168.1.49	TCP

Zdroj: vlastní zpracování

Obrázek 13 Wireshark - ukázka TCP paketu

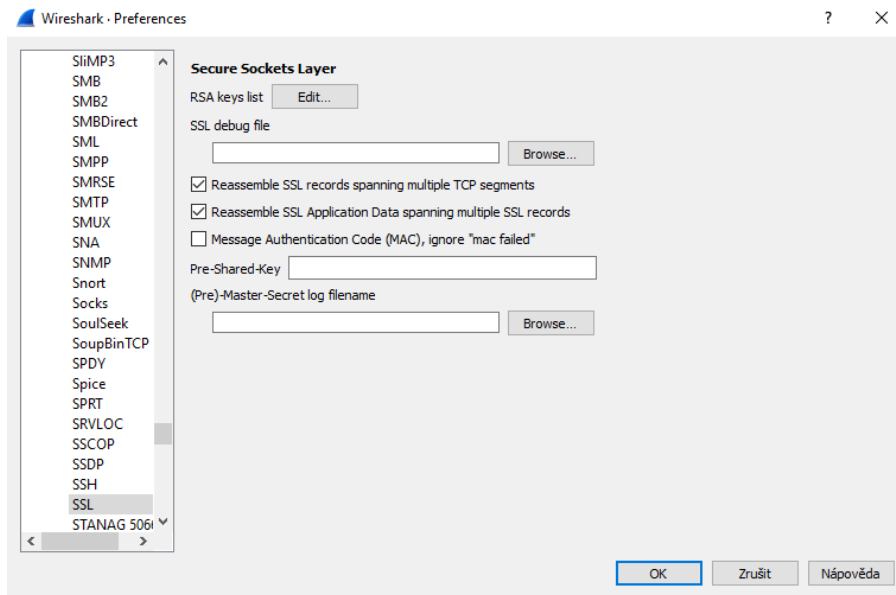
```
Transmission Control Protocol, Src Port: 8089, Dst Port: 37322, Seq: 3562, Ack: 7140, Len: 1253
  Source Port: 8089
  Destination Port: 37322
  [Stream index: 0]
  [TCP Segment Len: 1253]
  Sequence number: 3562 (relative sequence number)
  [Next sequence number: 4815 (relative sequence number)]
  Acknowledgment number: 7140 (relative ack number)
  1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x018 (PSH, ACK)
  Window size value: 363
  [Calculated window size: 46464]
  [Window size scaling factor: 128]
  Checksum: 0x8861 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [SEQ/ACK analysis]
  TCP payload (1253 bytes)
  Data (1253 bytes)
  Data: 17030304e03fd7a34b3e01cbd8be4c4c226524b37f0b6afa...
  Text [truncated]: \027\003\003\004\357\277\275?\357\277\275\357\277\275K>\001\357\277\275\357
  [Length: 1253]
```

Zdroj: vlastní zpracování

Veškerá komunikace probíhá skrze Secure WebSocket, který využívá transportní protokol TCP. SIP zprávy jsou zašifrovány a přenáší se jako data TCP streamu. Pro dešifrování komunikace je třeba nainportovat do programu Wireshark privátní klíč, který bude použit pro dešifrování. Dále je nutné definovat, pod kterým portem a jako který protokol komunikaci dešifrovat.

a) V programu Wireshark přejdeme do *Edit -> Preferences... -> Protocols -> SSL*

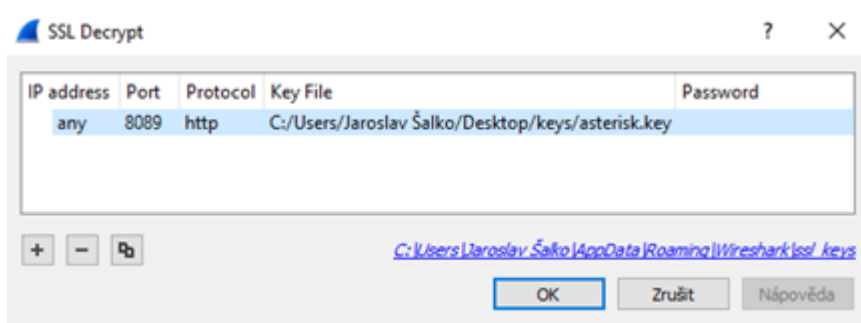
Obrázek 14 Wireshark - import klíče pro dešifrování 1



Zdroj: vlastní zpracování

b) Zde klikneme na tlačítko „*Edit...*“ u RSA keys list a vyplníme dle přiloženého **Obrázek 15.**

Obrázek 15 Wireshark - import klíče pro dešifrování 2



Zdroj: vlastní zpracování

- IP address

Zadáme IP adresu serveru/klienta, mezi kterými probíhá komunikace, kterou chceme dešifrovat. V tomto případě je Wireshark benevolentní a můžeme vepsáním slova „*any*“ ponechat filtr otevřený nebo jej nevyplňovat vůbec.

- Port

Je třeba definovat port na kterém má probíhat komunikace, kterou chceme dešifrovat. Vyplníme statický port na straně serveru. V našem případě se jedná o port 8089.

- Protocol

Jako protokol zadáme „*http*“. Jak bylo zmíněno v teoretickém úvodu, navázání WebSocket spojení je interpretováno jako požadavky na HTTP upgrade.

- Key File

Tato kolonka slouží pro zadání cesty k soukromému klíči. Vyberte již dříve vygenerovaný „*asterisk.key*“

- Password

Tato kolonka slouží pro vepsání před-sdíleného klíče, pakliže by byl klíč zašifrován. V našem případě necháme tuto kolonku nevyplněnou, neboť v souboru „*asterisk.key*“ se nachází soukromý klíč v nezašifrované podobě.

Nyní byste měli vidět dešifrované zprávy uvnitř WebSocketu.

Obrázek 16 Wireshark - dešifrovaná SIP komunikace

1524	67.645338	192.168.1.218	192.168.1.49	SIP/SDP
1526	67.646816	192.168.1.49	192.168.1.218	SIP
1532	67.853127	192.168.1.218	192.168.1.49	SIP
1537	67.895749	192.168.1.218	192.168.1.49	SIP/SDP
1539	67.898099	192.168.1.49	192.168.1.218	SIP
1544	67.905982	192.168.1.49	192.168.1.218	SIP/SDP
1556	68.125239	192.168.1.218	192.168.1.49	SIP
1569	68.428795	192.168.1.49	192.168.1.72	SIP/SDP
1577	68.553083	192.168.1.72	192.168.1.49	SIP
1578	68.592629	192.168.1.72	192.168.1.49	SIP

Zdroj: vlastní zpracování

Jak lze vidět ve zprávě typu „*Server hello*“, na **Obrázek 17**, při počátečním navazování spojení (TCP handshake), Asterisk server nabízí jako jedinou podporovanou šifru „*TLS_RSA_WITH_AES_128_CBC_SHA*“. Tudíž nevyužíváme pro výměnu klíčů šifry založené na ECDH algoritmu. Pokud by však z důvodů bezpečnosti protistrana tuto šifru nepodporovala, k navázání spojení by nedošlo.

Obrázek 17 Wireshark - tcp hand-shake, zpráva server hello

```

  ▾ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 49
    Version: TLS 1.2 (0x0303)
    > Random: e7cd40989713d1427069dbc8b856eaae83fb1ce12237c548...
    Session ID Length: 0
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Compression Method: null (0)
    Extensions Length: 9
    > Extension: renegotiation_info (len=1)
    > Extension: SessionTicket TLS (len=0)
  > TLSv1.2 Record Layer: Handshake Protocol: Certificate
  > TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done

```

Zdroj: vlastní zpracování

Další zajímavou zprávou je HTTP s požadavkem na upgrade, která je zobrazena na **Obrázek 18**. Tato zpráva se používá pro sestavení WebSocketu mezi klientem a serverem. Všimnout si můžeme, že aplikační data jsou označena jako „*http-over-tls*“. WebSocket se na straně serveru navazuje na portu 8089. To je definováno v popisu HTTP jako „*Host: 192.168.1.2:8089*“. V popisu samotného WebSocketu lze poté najít i definici vnořeného protokolu. „*Sec-WebSocket-protocol: sip*“. Tímto jsme si ověřili, že WebRTC využívá pro přenos signalizačních zpráv, protokol SIP tunelovaný v protokolu HTTP.

Obrázek 18 Wireshark - http-over-tls, http get-upgrade

```

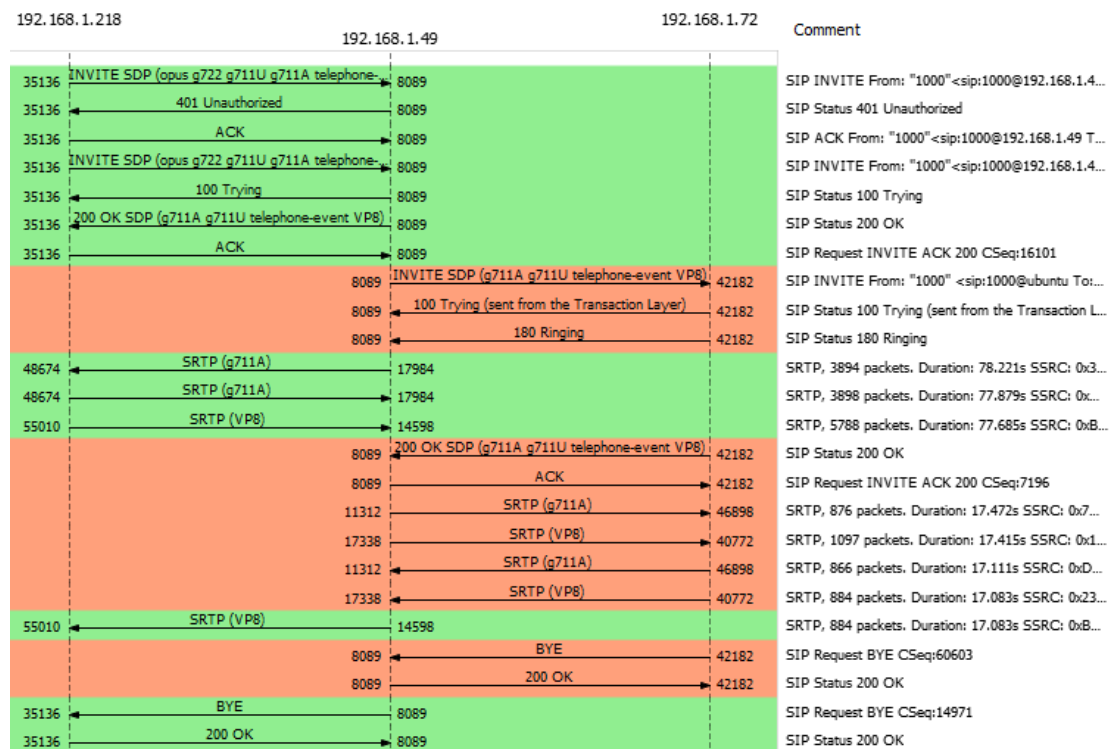
Hypertext Transfer Protocol
> GET /ws HTTP/1.1\r\n
Host: 192.168.1.49:8089\r\n
User-Agent: Mozilla/5.0 (Android 4.4.2; Mobile; rv:59.0) Gecko/59.0 Firefox/59.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: cs-CZ,cs;q=0.8,sk;q=0.6,en-US;q=0.4,en;q=0.2\r\n
Accept-Encoding: gzip, deflate, br\r\n
Sec-WebSocket-Version: 13\r\n
Origin: https://192.168.1.49\r\n
Sec-WebSocket-Protocol: sip\r\n
Sec-WebSocket-Extensions: permmessage-deflate\r\n
Sec-WebSocket-Key: 7UmnCGkPEQFdo/VI0YX8Gg==\r\n
Connection: keep-alive, Upgrade\r\n
Pragma: no-cache\r\n
Cache-Control: no-cache\r\n
Upgrade: websocket\r\n

```

Zdroj: vlastní zpracování

Poté co dojde k sestavení WebSocket spojení mezi serverem a klientem, jim již nadále provozujeme standardní SIP komunikaci, kterou si můžeme nechat i detailně zobrazit v nabídce *Telephony -> SIP Flows -> Flow Sequence*

Obrázek 19 Wireshark - SIP flow 1



Zdroj: vlastní zpracování

Zde si můžeme prohlédnout průběh SIP komunikaci mezi oběma WebRTC klienty a Asterisk serverem na portu 8089. SIP komunikace je zcela standardní. Samotný přenos RTP dat respektuje naše nastavení, které jsme prováděli v pjsip.conf a využívá zabezpečený SRTP, kdy jsme pro šifrování použili protokol DTLS s privátním klíčem „asterisk.key“. Pro přenos audia byl zvolen kodek g711a a pro video kodek VP8. Celý průběh testovacího hovoru je zachycen v souboru „asterisk.pcapng“, který se nachází v **Příloha E**.

11 Konfigurace WebRTC v PBX FreeSwitch

V této kapitole je popsána konfigurace WebRTC pro pobočkovou ústřednu FreeSWITCH v LTS verzi 1.6. Postup je psán krok za krokem tak, aby na jejím konci byl uživatel schopen provést úspěšný video-hovor skrze WebRTC klienta za pomoci PBX FreeSWITCH, jakožto signalizačního serveru. Veškerý zde uvedený postup byl proveden na virtualizovaném operačním systému Ubuntu 16.04. Použité příkazy a ukázky konfiguračních souborů jsou

psané takovou formou, aby je bylo možné použít metodou „*copy-paste*“. V případě, že nemáte PBX FreeSWITCH nainstalovanou, můžete využít instalační návod v **Příloha C**.

11.1 Konfigurace modulu Sofia, pro naslouchání WebSocket spojení

Modul Sofia zajišťuje v PBX FreeSWITCH konektivitu klientům s koncovými účty. Tato komunikace probíhá skrze protokol SIP. Sofia je poté v PBX FreeSWITCH obecný název pro libovolného klienta, který používá SIP protokol pro manipulaci s koncovým účtem. Pro podporu WebRTC je nutné povolit naslouchání WebSocket spojení modulu Sofia na příslušném portu. Pro konfiguraci modulu Sofia, kde provedeme toto nastavení, slouží konfigurační soubor `/usr/local/freeswitch/conf/sip_profiles/internal.xml`.

- a) V souboru `internal.xml`, v sekci `settings`, vyhledejte a odkomentujte, případně doplňte tyto řádky:

```
<param name="tls-cert-dir" value="/usr/local/freeswitch/certs"/> <param  
name="wss-binding" value=":7443"/>
```

Proved'te znovunačtení konfiguračních souborů, příkazem `reloadxml`, z konzole aplikace FreeSWITCH. Jestli aplikace FreeSWITCH naslouchá na portu 7443 se můžeme přesvědčit zadáním následujícího příkazu z konzole aplikace FreeSWITCH:

```
sofia status profile internal
```

Ve výstupu tohoto příkazu byste měli dohledat tento parametr:

```
WSS-BIND-URL  
sips:mod_sofia@<freeswitch_server_IP>:7443;transport=wss
```

11.2 Konfigurace uživatelských účtů

PBX FreeSWITCH poskytuje již v rámci instalace několik předkonfigurovaných koncových účtů. Tyto účty nalezneme v adresáři `/usr/local/freeswitch/conf/directory/default`. Každý účet je definovaný svým vlastním xml souborem. V těchto souborech můžeme konfigurovat parametry jako jsou uživatelské jméno, heslo, přidružený kontext, caller ID atd.

Pro naše účely není třeba v těchto souborech nic měnit. Pro následující postupy budeme využívat koncové účty 1000 a 1001.

```
<include>
  <user id="1000">
    <params>
      <param name="password" value="${default_password}"/>
      <param name="vm-password" value="1000"/>
    </params>
    <variables>
      <variable name="toll_allow" value="domestic,international,local"/>
      <variable name="accountcode" value="1000"/>
      <variable name="user_context" value="default"/>
      <variable name="effective_caller_id_name" value="Extension 1000"/>
      <variable name="effective_caller_id_number" value="1000"/>
      <variable name="outbound_caller_id_name"
value="${outbound_caller_name}"/>
      <variable name="outbound_caller_id_number"
value="${outbound_caller_id}"/>
      <variable name="callgroup" value="techsupport"/>
    </variables>
  </user>
</include>
```

11.3 Konfigurace základního kontextu

PBX FreeSWITCH poskytuje již v rámci instalace základní kontext s názvem „*default*“. Směrovací pravidla tohoto kontextu jsou definována v konfiguračním souboru */usr/local/freeswitch/conf/dialplan/default.xml*. Tento kontext obsahuje základní směrovací pravidla a umožňuje směrování mezi předkonfigurovanými koncovými účty. Pro naše účely budeme využívat volání na číslo 9196, které dle vstupních směrovacích pravidel poskytuje volanému echo v reálném čase, viz. ukázka níže a volání na koncové účty 1000 a 1001:

```
<extension name="echo">
  <condition field="destination_number" expression="^9196$">
    <action application="answer"/>
    <action application="echo"/>
  </condition>
</extension>
```

```
</condition>
</extension>
```

11.4 Příprava prohlížeče

Pro přípravu prohlížeče použijeme stejný postup jako v případě PBX Asterisk viz. kapitola 10.7. Lišit se bude jen nepatrně příkaz pro konverzi certifikátu, do formátu podporovaného internetovými prohlížeči.

a) Přejdeme do adresáře, kde se nachází vygenerované klíče a certifikáty

```
cd /usr/local/freeswitch/certs
```

b) V tomto adresáři použijeme příkaz nástroje OpenSSL pro konverzi mezi formáty

```
openssl pkcs12 -export -out <název_výstupního_certifikátu>.p12 -inkey
wss.key -in wss.crt -certfile ca.crt
```

Import certifikátu do prohlížeče poté provedete totožně, jak tomu bylo v případě kapitoly 10.7.2 .

11.5 Testovací hovor

Přejdeme na stránky našeho Apache serveru s WebRTC klientem, kterému jsme se věnovali v kapitole 9. WebRTC klienta nakonfigurujeme dle následujícího postupu. V kolonce „*Registration*“ vyplníme následující údaje:

- Display Name: 1000
- Private Identity*: 1000
- Public Identity*: sips:1000@<IP_FreeSWITCH_serveru>
- Password: 1234
- Realm*: <IP_FreeSWITCH_serveru>

Poté přejdeme do expert módu, kliknutím na tlačítko „*Expert mode?*“ a specifikujeme náš WebSocket pro transport. Kolonku „*WebSocket Server URL*“ vyplňte následujícím způsobem:

- WebSocket Server URL: wss://<IP_freeswitch_serveru>:7443/ws

Nastavení uložíme kliknutím na tlačítko „Save“ a vrátíme se na stránku s oknem registrace. Při stisknutí tlačítka „LogIn“, bychom měli být nahoře informováni o úspěšném zaregistrování statusem „Connected“.

Přesuneme se do konzole aplikace FreeSWITCH a necháme si zobrazit informace o všech přihlášených uživateli následujícím příkazem:

```
sofia status profile internal reg
```

Ve výpise příkazu můžeme vyčíst, zda je klient dostupný, z jaké IP adresy a portu je přihlášený, ke kterému uživatelskému účtu, případně další informace, jak je ukázáno na příkladu níže pro uživatele 1000.

```
Registrations:=====
=Call-ID:      65684810-86f2-aa13-485b-b1d5e3d46345
User:       1000@192.168.1.119
Contact:       "1000" sips:1000@df7jal23ls0d.invalid;rtcweb-
breaker=no;transport=wss;fs_nat=yes;fs_path=sips%3A1000%40192.168.70.148%3A53358%3Brtcweb-
breaker%3Dno%3Btransport%3Dwss
Agent:      IM-client/OMA1.0 sipML5-v1.2016.03.04
Status:        Registered(TLS-NAT)(unknown) EXP(2018-05-14 01:36:11) EXPSECS(244)
Ping-Status: Reachable
Ping-Time:     0.00
Host:          ubuntu
IP:         192.168.1.72
Port:       53358
Auth-User:     1000
Auth-Realm:    192.168.1.119
MWI-Account:   1000@192.168.1.119

Total items returned: 1
=====
```

Z výpisu například vidíme, že klient Sipml5 je přihlášený z IP 192.168.1.72 k uživatelskému účtu 1000 a používá transportní metodu wss.

- a) Stejným způsobem zaregistrujte z jiného zařízení WebRTC klienta k účtu 1001.
- b) Provedeme testovací video-hovor mezi klienty 1000 a 1001, pomocí ovládacího okna Sipml5 klienta. Otevřeme si nabídku „Call“ a vybereme „Video“.
- c) Budeme dotázáni, zda chceme dané stránce/aplikaci povolit přístup k mikrofonu. Povolení přidělíme kliknutím na tlačítko „Allow“.

- d) Následně bychom měli v ovládacím okně Sipml5 klienta vidět v levém horním rohu status „*Call in progress*“
- e) Při úspěšném sestavení SIP komunikace, bude status odpovídat stavu „*In Call*“.
V tomto stavu již dochází k RTP přenosu.

V konzoli aplikace FreeSWITCH byste měli vidět podrobný průběh hovoru. Ten by měl být podobný níže uvedené ukázce.

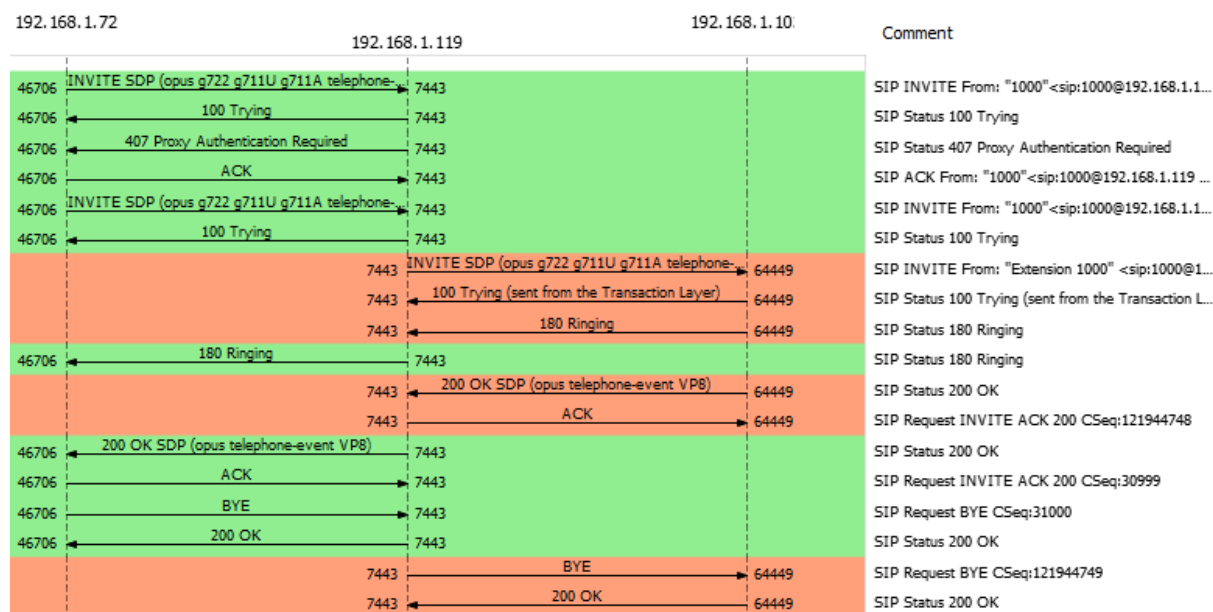
```
2018-05-14 03:19:34.858778 [NOTICE] switch_channel.c:1104 New Channel
sofia/internal/1000@192.168.1.119 [4c9d5a92-5760-11e8-93da-63b589367e3e]
2018-05-14 03:19:34.957992 [INFO] mod_dialplan_xml.c:637 Processing 1000 <1000>->1001 in
context default
2018-05-14 03:19:45.338046 [INFO] switch_ivr_async.c:4173 Bound B-Leg: *1
execute_extension::dx XML features
2018-05-14 03:19:45.338046 [INFO] switch_ivr_async.c:4173 Bound B-Leg: *2
record_session::/usr/local/freeswitch/recordings/1000.2018-05-14-03-19-45.wav
2018-05-14 03:19:45.338046 [INFO] switch_ivr_async.c:4173 Bound B-Leg: *3
execute_extension::cf XML features
2018-05-14 03:19:45.338046 [INFO] switch_ivr_async.c:4173 Bound B-Leg: *4
execute_extension::att_xfer XML features
2018-05-14 03:19:45.358282 [NOTICE] switch_channel.c:1104 New Channel
sofia/internal/1001@df7jal23ls0d.invalid [52df9c08-5760-11e8-93f9-63b589367e3e]
2018-05-14 03:19:45.898041 [NOTICE] sofia.c:7192 Ring-Ready
sofia/internal/1001@df7jal23ls0d.invalid!
2018-05-14 03:19:45.898041 [NOTICE] mod_sofia.c:2273 Ring-Ready
sofia/internal/1000@192.168.1.119!
2018-05-14 03:20:00.978712 [NOTICE] sofia.c:8180 Channel
[sofia/internal/1001@df7jal23ls0d.invalid] has been answered
2018-05-14 03:20:00.997974 [INFO] switch_rtp.c:3581 Activate RTP/RTCP audio DTLS server
2018-05-14 03:20:00.997974 [INFO] switch_rtp.c:3730 Changing audio DTLS state from OFF to
HANDSHAKE
2018-05-14 03:20:00.997974 [INFO] switch_core_media.c:7478 Activating VIDEO RTCP PORT 0
interval 1000 mux 1
2018-05-14 03:20:01.017897 [NOTICE] switch_ivr_originate.c:3647 Channel
[sofia/internal/1000@192.168.1.119] has been answered
2018-05-14 03:20:09.978141 [NOTICE] sofia.c:1012 Hangup
sofia/internal/1001@df7jal23ls0d.invalid [CS_EXCHANGE_MEDIA] [NORMAL_CLEARING]
2018-05-14 03:20:11.028285 [NOTICE] switch_ivr_bridge.c:1764 Hangup
sofia/internal/1000@192.168.1.119 [CS_EXECUTE] [NORMAL_CLEARING]
2018-05-14 03:20:11.028285 [NOTICE] switch_core_session.c:1683 Session 2
(sofia/internal/1001@df7jal23ls0d.invalid) Ended
2018-05-14 03:20:11.028285 [NOTICE] switch_core_session.c:1687 Close Channel
sofia/internal/1001@df7jal23ls0d.invalid [CS_DESTROY]
2018-05-14 03:20:11.038264 [NOTICE] switch_core_session.c:1683 Session 1
(sofia/internal/1000@192.168.1.119) Ended
2018-05-14 03:20:11.038264 [NOTICE] switch_core_session.c:1687 Close Channel
sofia/internal/1000@192.168.1.119 [CS_DESTROY]
```


Z konzole můžeme vyčíst, že modul `mod_dialplan_xml`, zpracovává požadavek uživatele 1000, který „volá“ pravidlo 1001 v kontextu „*default*“. Ve výpisu je také patrné, že pro přenos RTP/RTCP, byl použit zabezpečovací transportní protokol DTLS. Dále si můžeme ještě všimnout otevření samostatného portu pro RTCP pro video.

11.6 Zachycení a analýza komunikace v programu Wireshark

Zachycená komunikace je opět celá skryta v transportním protokolu TCP. Pro dešifrování je třeba do programu Wireshark naimportovat příslušný klíč, který byl použit pro šifrování komunikace. Pro dešifrování aplikujte stejný postup, který je podrobně popsán v kapitole 10.6. Na **Obrázek 20** je ukázka již dešifrované standardní SIP komunikace, zachycující video-hovor mezi WebRTC klienty s registrovanými účty 1000 a 1001.

Obrázek 20 Wireshark - SIP flow 2



Zdroj: vlastní zpracování

Na **Obrázek 20** je zachycena SIP komunikace mezi oběma WebRTC klienty a FreeSWITCH serverem na portu 7443. SIP komunikace je zcela standardní. Jako kodek pro audio zde byl použit opus, který je integrován v PBX FreeSWITCH již při instalaci a měl by poskytnout vyšší kvalitu hovorů, nežli klasické g711a. Pro video byl opět použit kodek VP8. Průběh kompletního testovacího hovoru je zachycen v souboru „*freeswitch.pcap*“, který se nachází v **Příloha E**.

12 Laboratorní úloha - zadání

Praktická část této práce je věnovaná tvorbě laboratorní úlohy pro studenty oboru *telekomunikační a informační systémy*. Úloha má především seznámit studenty s WebRTC v PBX Asterisk.

Úloha v úvodu obsahuje krátkou teorii, která představí studentům WebRTC a seznámí je s nutnými pojmy, které je třeba znát, pro pochopení následující konfigurace. Na teorii navazují okomentované ukázky konfiguračních souborů, již upravené pro podporu WebRTC. Na těchto ukázkách je studentům vysvětleno co, kde a jak je třeba nakonfigurovat, pro úspěšné zprovoznění podpory pro WebRTC. Samotné zadání úlohy je rozděleno na devět úkolů.

První úkol souvisí se studiem teorie v úvodu laboratorní úlohy.

Druhý úkol je zaměřený na konfiguraci vestavěného HTTP serveru, nutného pro zpracování HTTP požadavků. Studenti HTTP server nakonfigurují a před pokračováním v úloze ověří, zda server naslouchá na příslušných portech.

Třetí úkol se věnuje konfiguraci koncových účtů. Studenti rozšíří dle návodu konfiguraci ukázkového účtu tak, aby podporoval připojení k WebRTC klientům. Stejným způsobem poté přidají i účty nové.

Ve čtvrtém úkolu studenti upraví směrovací pravidla, aby bylo možné v následujících úkolech provádět testovací hovory a ověřovat tak funkčnost nastavení.

V pátém úkolu si studenti zaregistrují vytvořené účty na VoIP telefon a na WebRTC klienta v desktopovém internetovém prohlížeči. Provedou testovací hovor, kterým si ověří doposud provedené zásahy do konfigurace ústředny. Průběh hovoru si zobrazí v konzoli ústředny.

V šestém úkolu si studenti nainstalují certifikát místní, známe certifikační autority do svého smartphonu s operačním systémem android. Zaregistrují si účet k WebRTC klientovi z mobilního internetového prohlížeče Firefox a opět provedou testovací hovor.

Sedmí úkol je zaměřen na konfiguraci SIP trunku k nadřazené ústředně, aby se bylo možno dovolat na ostatní pracoviště v laboratoři.

V osmém úkolu studenti opět upraví směrovací pravidla tak, aby za využití trunku z předešlého úkolu, byli schopni volat na ostatní pracoviště. Správnost konfigurace si ověří realizací video-hovoru, za pomoci svých mobilních zařízení, s kolegy na ostatních pracovištích.

Devátý, poslední úkol, je věnovaný analýze hovoru mezi WebRTC klienty, zachyceného pomoci paketového analyzátoru Wireshark. Protože je WebRTC komunikace šifrovaná, budou muset studenti podle návodu hovor nejdříve dešifrovat. Na dešifrované komunikaci provedou analýzu hovoru. Tato analýza slouží jako demonstrace teorie v úvodu, kdy se studenti přesvědčí, že WebRTC využívá tunelování SIP protokolu v protokolu HTTP.

12.1 Laboratorní úloha – praktická část

Samotná laboratorní úloha se nachází v **Příloha D**.

Závěr

V úvodu této práce byl představen projekt WebRTC. Krátce jsem se zmínil o jeho historii, a především jsem se snažil poukázat na jeho přednost v podobě absence rozšiřujících plug-inů, které jsou u konkurence nutné pro poskytnutí plné real-time komunikace, včetně zvuku a videa, v rámci internetového prohlížeče. Dozvěděli jsme se, že WebRTC sdružuje celou sadu protokolů a mechanismů, které jsou implementovány přímo v prohlížeči pomocí JavaScript kódu a jazyka HTML5. Nad samotným kódem byly postaveny a standardizovány API rozhraní, skrze které se ke kódu přistupuje. Navzdory tomu, jak je či není kód složitý, vývojáři webových aplikací tedy pouze využívají konkrétní API rozhraní. Následující kapitoly byly věnovány nejdůležitějším API rozhraním, zajišťující funkcionalitu WebRTC klientů. Jmenovitě se jedná o API rozhraní `RTCPeerConnection`, zajišťující navázání spojení, `MediaStream`, poskytující přístup k HW zdrojům mediálních dat a `RTCDataChannel`, který zajišťuje transport mediálních dat mezi koncovými body WebRTC komunikace. V návaznosti na předchozí kapitoly jsou dále uvedeny a popsány použité protokoly a mechanismy, s kterými WebRTC pracuje a které jsou nezbytné pro zajištění tohoto druhu komunikace.

Poté co je čtenář seznámen se skladbou WebRTC aplikací v prohlížeči, pokračuje práce principem komunikace mezi WebRTC klienty, kdy jsem se zaměřil především na počáteční navázání spojení a signalizaci. Zde se také poprvé dostáváme k PBX ústřednám, které jsou nezbytné pro zprostředkování signalizace. WebRTC samotné totiž blíže nespecifikuje signalizační protokol a také nezajišťuje distribuci veřejné identifikace WebRTC klientů na síti. Proto se dostáváme k PBX ústřednám, sloužící jako signalizační servery. Další část práce popisuje způsob, kterým PBX Asterisk, FreeSWITCH a Kamailio integrují podporu WebRTC.

PBX Asterisk implementuje WebRTC za použití integrovaného HTTP serveru a transportní metody SIP over WebSocket. Pomocí této transportní metody dochází k tunelování standardního signalizačního SIP protokolu v protokolu HTTP. Asterisk v testované LTS verzi 13 podporuje pouze audio a videohovory typu point-to-point. Podpora videokonferencí přichází až ve verzi 15. Vývojáři nicméně chystají novou verzi ústředny, Asterisk 16, která by měla být nástupcem LTS verze Asterisk 13. Pro PBX Asterisk jsem i vytvořil návod, který popisuje všechny nezbytné kroky pro konfiguraci WebRTC.

PBX FreeSWITCH v LTS verzi 1.6 implementuje WebRTC za použití rozšiřujících modulů `mod_httapi` a `mod_http_cache`. Tyto moduly slouží pro zpracování HTTP požadavků. Nejedná se však přímo o HTTP server, jako v případě PBX Asterisk. Jako signalizační protokol pro WebRTC používá PBX FreeSWITCH protokol SIP, případně vlastní protokol Verto. V případě použití protokolu SIP FreeSWITCH podporuje audio a videohovory typu point-to-point. V případě použití vlastního protokolu Verto, s použitím Verto komunikátorů, je možné realizovat i videokonferenční hovory. Pro implementaci WebRTC za použití signalizačního protokolu SIP, je v této práci vytvořen návod se všemi potřebnými kroky.

PBX Kamailio LTS verze své ústředny nevydává. Její poslední Stable verze je verze 5.1.3 z května roku 2018. Kamailio jako takové přímo WebRTC nepodporuje, neboť jeho hlavním zaměřením je fungování jako výkonného SIP proxy serveru. Ze signalizačního hlediska však je schopen s WebRTC pracovat, neboť obsahuje od verze 4.0.0 vestavěný HTTP server a modul WEBSOCKET, který mu umožňuje sestavovat WebSocket spojení. Pro plnohodnotnou podporu WebRTC musí být Kamailio využíváno v kombinaci i s jinými projekty, jako například RTPEngine, PBX Asterisk, FreeSWITCH a jiné, které WebRTC podporují. U PBX Kamailio se mi bohužel nepodařilo úspěšně definovat chování WebSocketu, a tak pro tuto ústřednu návod chybí.

V praktické části této práce jsem navrhl laboratorní úlohu pro studenty předmětu *telekomunikační a informační systémy*. Tato úloha provede studenty konfigurací WebRTC pro PBX Asterisk. Po úspěšné konfiguraci si budou studenti schopni zprostředkovat pomocí WebRTC klienta Sipml5 videohovor, za použití svých vlastních smartphonů s operačním systémem android. V závěru úlohy si studenti za pomoci paketového analyzátoru Wireshark odchyť WebRTC komunikaci. Zachycenou komunikaci dešifrují a následně analyzují, aby se přesvědčili, že pro signalizaci používá WebRTC standardní SIP protokol, tunelovaný v protokolu HTTP.

Seznam použitých zdrojů

- [1] Asterisk Ready To Get Started With Asterisk? [online]. [cit. 2018-05-20]. Dostupné z: <https://www.asterisk.org/>
- [2] Asterisk 13 Configuration_res_pjsip. *Asterisk* [online]. [cit. 2017-12-13]. Dostupné z: https://wiki.asterisk.org/wiki/display/AST/Asterisk+13+Configuration_res_pjsip
- [3] Asterisk Builtin mini-HTTP Server. *Asterisk* [online]. [cit. 2017-11-24]. Dostupné z: <https://wiki.asterisk.org/wiki/display/AST/Asterisk+Builtin+mini-HTTP+Server>
- [4] *Asterisk Forums* [online]. [cit. 2017-12-13]. Dostupné z: <http://forums.asterisk.org>
- [5] BROWSER APIS AND PROTOCOLS. *High Performance Browser Networking* [online]. [cit. 2017-11-17]. Dostupné z: <https://hpbnc.co/webrtc/>
- [6] *Cipherli.st* [online]. [cit. 2017-12-04]. Dostupné z: <https://cipherli.st/>
- [7] DUNKLEY Peter, KAMAILIO – WebSocket Module documentation [online]. [cit. 2018-05-17] Dostupné z: <http://www.kamailio.org/docs/modules/5.1.x/modules/websocket.html2>
- [8] Frequent Questions. *WebRTC* [online]. [cit. 2017-11-04]. Dostupné z: <https://webrtc.org/faq/>
- [9] *FreeSWITCH* [online]. [cit. 2017-12-13]. Dostupné z: <https://freeswitch.org/>
- [10] Getting Started WebRTC - AstriCon 2014. *Youtube* [online]. [cit. 2017-12-02]. Dostupné z: https://www.youtube.com/watch?v=btfX884mK_c&t=708s
- [11] How to Install Asterisk 13 on Ubuntu 16.04 from Source. *LinOxide* [online]. [cit. 2017-11-08]. Dostupné z: <https://linoxide.com/ubuntu-how-to/install-asterisk-13-ubuntu-16-04-source/>
- [12] How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 16.04. *DigitalOcean* [online]. [cit. 2017-12-13]. Dostupné z: <https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-apache-in-ubuntu-16-04>
- [13] How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 16.04. *DigitalOcean* [online]. [cit. 2017-12-13]. Dostupné z: <https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04>
- [14] Implementation Lessons using WebRTC in Asterisk. *Youtube* [online]. [cit. 2017-11-08]. Dostupné z: <https://www.youtube.com/watch?v=GHFduPTNE1Q>

- [15] Introduction to WebRTC protocols. *Webová dokumentace MDN* [online]. [cit. 2017-11-17]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols
- [16] *Kamailio* [online]. [cit. 2017-12-05]. Dostupné z: <http://kamailio.org/>
- [17] Media Capture and Streams. *World Wide Web Consortium* [online]. [cit. 2017-11-04]. Dostupné z: <https://w3c.github.io/mediacapture-main/>
- [18] PJSIP debug TLS issues with Wireshark. *Asterisk Community* [online]. [cit. 2017-10-31]. Dostupné z: <https://community.asterisk.org/t/pjsip-debug-tls-issues-with-wireshark/70230>
- [19] SIMPL5. *GitHub* [online]. [cit. 2017-12-13]. Dostupné z: <https://github.com/DoubangoTelecom/sipml5>
- [20] WebRTC 1.0: Real-time Communication Between Browsers. *World Wide Web Consortium* [online]. [cit. 2017-12-01]. Dostupné z: <https://www.w3.org/TR/webrtc/>
- [21] WebRTC About & FAQ [online]. [cit. 2018-05-20]. Dostupné z: <https://webrtc.org/faq/>
- [22] WebRTC. *FreeSWITCH* [online]. [cit. 2017-12-05]. Dostupné z: <https://freeswitch.org/confluence/display/FREESWITCH/WebRTC>
- [23] WebRTC How it Works and How it Breaks. *Youtube* [online]. [cit. 2017-12-13]. Dostupné z: <https://www.youtube.com/watch?v=3TbVi9aB09k&t=546s>
- [24] WebRTC tutorial using SIPML5. *Asterisk* [online]. [cit. 2017-10-31]. Dostupné z: <https://wiki.asterisk.org/wiki/display/AST/WebRTC+tutorial+using+SIPML5>

Seznam příloh

Příloha A	Instalace Apache serveru
Příloha B	Instalace pobočkové ústředny Asterisk 13
Příloha C	Instalace pobočkové ústředny FreeSwitch 1.6
Příloha D	Laboratorní úloha 6 - PBX Asterisk – WebRTC
Příloha E	CD/DVD s konfiguračními soubory

Příloha A

A.1 Instalace Apache serveru

V této příloze je popsána instalace webového Apache serveru s vlastně podepsanými SSL certifikáty. Veškerý, zde uvedený postup byl proveden na virtualizovaném operačním systému Ubuntu 16.04. Použité příkazy jsou psané takovou formou, aby je bylo možné použít metodou „copy-paste“. Při dodržení postupu budete mít na konci kapitoly nainstalován Apache server se všemi důležitými predispozicemi pro integraci WebRTC klienta, které se věnuje diplomová práce v kapitole 9.

- a) Poté co se přihlásíte na váš Ubuntu OS, je třeba zkontrolovat a případně stáhnout aktualizace balíčků, které OS Ubuntu používá.

```
sudo apt-get update
```

- b) Nainstalujeme webový HTTP Apache server

```
sudo apt-get install apache2
```

- c) Po nainstalování Apache, přidáme do souboru `/etc/apache2/apache2.conf` položku `ServerName`. Pokud nemáme k serveru přiřazen doménový název, můžeme použít IP adresu našeho serveru. Na konec souboru přidáme následující řádek:

```
ServerName <server_domain_or_IP>
```

- d) Restartujeme apache, aby se projevíly změny, které jsme provedli.

```
sudo systemctl restart apache2
```

- e) Pokud postupujete podle pokynů této práce na nově nainstalovaném OS Ubuntu, neměli byste nijak blokovat http nebo https provoz na bráně firewall svého serveru. Můžete se ujistit, že nijak neblokujete tento provoz zadáním příkazu:

```
sudo ufw app list
```

Měli byste dostat tento výstup:

```
Available applications:
  Apache
  Apache Full
```

Pokud se podíváme na profil Apache Full, měl by ukazovat, že umožňuje provoz na portech 80 a 443.

```
sudo ufw app info "Apache Full"
```

Výstup:

```
Profile: Apache Full
Title: Web Server (HTTP,HTTPS)
Description: Apache v2 is the next generation of the omnipresent Apache
web server.
Ports:
80,443/tcp
```

f) Povolíme příchozí provoz pro tento profil.

```
sudo ufw allow in "Apache Full"
```

Nyní můžete okamžitě provést kontrolu vašeho serveru. Otevřete libovolný prohlížeč a navštivte adresu http://<your_server_FQDN_or_IP_address>. Měla by se vám načíst úvodní stránka Apache2 Ubuntu web serveru.

A.2 Vygenerování vlastního SSL certifikátu

TLS/SSL funguje pomocí kombinace veřejného certifikátu a soukromého klíče. Soukromí klíč je uložen na straně serveru a je třeba jej uchovat v tajnosti. Tento klíč se používá k šifrování obsahu odeslaného klientům. SSL certifikát je veřejně sdílený s každým, kdo požaduje obsah serveru. Tento certifikát, který obsahuje veřejný klíč se používá na straně klienta k dešifrování obsahu podepsaného přidruženým tajným klíčem.

a) Vytvoření vlastně podepsaného certifikátu se soukromím klíčem provedeme pomocí nástroje OpenSSL, zadáním následujícího příkazu:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout
etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-
selfsigned.crt
```

- openssl - toto je základní příkazový nástroj pro vytváření a správu certifikátů, klíčů a dalších OpenSSL souborů;
- req - Tento podřízený příkaz specifikuje, že chceme použít standard X.509 pro vytvoření infrastruktury, dle níž se vytvoří dvojice soukromého a veřejného klíče;
- -x509 - Tímto dále rozvíjíme podřízený příkaz req a specifikujeme, že chceme vytvořit podepsaný certifikát namísto generování požadavku na podepsání certifikátu, jak tomu obvykle bývá;
- -nodes - Tento podřízený příkaz zajistí přeskočení přístupové fáze. Využíváme jej proto, aby náš Apache server mohl číst soubor bez zásahu uživatele. Tím zamezíme vynucení ověření pomocí hesla pokaždé, kdy restartujeme Apache server;
- -days 365 - zde definujeme dobu, po kterou bude námi vygenerovaný certifikát považován za platný;
- -newkey rsa:1024 - Toto určuje, že chceme spolu s certifikátem vytvořit i nový klíč potřebný pro podepsání certifikátu. Tento klíč bude vytvořen pomocí kryptografického algoritmu RSA a bude mít délku 1024 bitů;
- -keyout - Tímto podřízeným příkazem specifikujeme, kam se umístí vytvořený soukromý klíč;
- -out - Tímto podřízeným příkazem specifikujeme, kam se umístí vytvořený certifikát.

Po zadání tohoto příkazu budete dotázáni na řadu otázek ohledně vašeho serveru. Vaše odpovědi se použijí při vytváření certifikátu. Zde máte ukázkou, jak by měly vypadat vaše odpovědi.

Výstup:

```
Country Name (2 letter code) [AU]:CZ
State or Province Name (full name) [Some-State]:Czech republic
Locality Name (eg, city) []:Brno
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Test1
Organization Unit Name (eg, section) []:Test2
Common Name (e.g. server FQDN or YOUR name) []:<server_IP_address>
Email Address []:admin@<your_domain>.com
```

Oba soubory budou vytvořeny do adresářů zadaných v příkazu. Certifikát tedy naleznete v adresáři `/etc/ssl/certs/` pod názvem „`apache-selfsigned.crt`“. Soukromý klíč s názvem „`apache-selfsigned.key`“ bude umístěn v adresáři `/etc/ssl/private/`.

- a) Dále vytvoříme skupinu parametrů, která poslouží pro vytvoření Diffie-Hellman algoritmu, který se používá při vyjednávání parametrů během navazování spojení.

```
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 1024
```

A.3 Konfigurace Apache pro použití vlastně podepsaného certifikátu

V přechodím kroku jsme vytvořili vlastně podepsaný certifikát a soukromí klíč. Nyní stačí upravit konfiguraci Apache serveru tak, abychom tento certifikát využili.

- a) Nejprve vytvoříme konfigurační fragment a definujeme určitá nastavení protokolu SSL. Vytvoření nového fragmentu provedeme v adresáři `/etc/apache2/conf-available/`. Vytvořený soubor pojmenujeme „`ssl-param.conf`“.
- b) Obsah tohoto souboru definuje, jak bude použit náš certifikát pro protokol SSL. Konfigurace pochází se serveru `https://cipherli.st/`. Zkopírujte níže uvedené řádky do souboru `ssl-param.conf`.

```
# from https://cipherli.st/
# and https://raymii.org/s/tutorials/Strong_SSL_Security_On_Apache2.html

SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
SSLProtocol All -SSLv2 -SSLv3
SSLHonorCipherOrder On
```

```
# Disable preloading HSTS for now.  You can use the commented out header
line that includes
# the "preload" directive if you understand the implications.
#Header always set Strict-Transport-Security "max-age=63072000;
includeSubdomains; preload"
Header always set Strict-Transport-Security "max-age=63072000;
includeSubdomains"
Header always set X-Frame-Options DENY
Header always set X-Content-Type-Options nosniff
# Requires Apache >= 2.4
SSLCompression off
SSLSessionTickets Off
SSLUseStapling on
SSLStaplingCache "shmcb:logs/stapling-cache(150000)"

SSLOpenSSLConfCmd DHParameters "/etc/ssl/certs/dhparam.pem"
```

- c) Dále upravíme soubor */etc/apache2/sites-available/default-ssl.conf* tak, aby využíval námi vygenerovaný certifikát a soukromý klíč.

```
echo "" > /etc/apache2/sites-available/default-ssl.conf
```

- d) Do otevřeného souboru zkopírujte tuto konfiguraci

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerAdmin admin@freeswitch.com
        ServerName 192.168.2.2

        DocumentRoot /var/www/html

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        SSLEngine on

        SSLCertificateFile      /etc/ssl/certs/apache-
selfsigned.crt
        SSLCertificateKeyFile  /etc/ssl/private/apache-
selfsigned.key
```

```
<FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars
</FilesMatch>
<Directory /usr/lib/cgi-bin>
    SSLOptions +StdEnvVars
</Directory>

BrowserMatch "MSIE [2-6]" \
    nokeepalive ssl-unclean-shutdown \
    downgrade-1.0 force-response-1.0

</VirtualHost>
</IfModule>
```

e) Pro aplikování těchto změn, musíme povolit SSL moduly a restartovat službu Apache.

```
sudo a2enmod ssl && a2enmod headers && a2ensite default-ssl && a2enconf
ssl-params && apache2ctl configtest sudo a2enmod headers
```

```
sudo systemctl restart apache2
```

Příloha B

B.1 Instalace pobočkové ústředny Asterisk 13

V této příloze je popsána instalace pobočkové ústředny Asterisk v LTS verzi 13 se všemi potřebnými predispozicemi pro následnou konfiguraci WebRTC. Pro konfiguraci WebRTC je zapotřebí, aby Asterisk server obsahoval moduly `res_crypto`, `res_http_websocket` a `res_pjsip_transport_websocket`. Na konci této kapitoly, by měl být čtenář schopen provést za pomoci PBX Asterisk, jakožto signalizačního serveru, úspěšný testovací hovor s využitím WebRTC klienta.

- a) Poté co se přihlásíte na váš Ubuntu server je třeba se přepnout na uživatele s právy `root`.

```
sudo su
```

- b) Přesuneme se do složky, kde budeme kompilovat Asterisk.

```
cd /usr/src
```

- c) Stáhneme Asterisk.

```
Wget downloads.asterisk.org/pub/telephony/asterisk/asterisk-13-  
current.tar.gz
```

- d) Vyextrahujeme balíček.

```
tar zxvf asterisk-13-current.tar.gz
```

- e) Vstoupíme do složky Asterisk z vyextrahovaného balíčku.

```
cd asterisk-13*
```

- f) Předtím, než zahájíme kompilaci Asterisku, stáhneme a zkompilujeme PJPROJECT, což není, nic jiného, než balíček obsahující podporu PJSIP.

```
git clone git://github.com/asterisk/pjproject pjproject
```

```
cd pjproject
```

```
./configure --prefix=/usr --enable-shared --disable-sound --disable-resample --disable-video --disable-opencore-amr CFLAGS='-O2 -DNDEBUG'
```

```
make dep
```

```
make
```

```
make install
```

```
ldconfig
```

```
ldconfig -p |grep pj
```

```
cd ..
```

g) Nyní se můžeme pustit do kompilace Asterisku

```
contrib/scripts/get_mp3_source.sh
```

```
contrib/scripts/install_prereq install
```

```
ITU-T telephone code: 420
```

h) Pokračujeme základní konfigurací Asterisku

```
./configure
```

```
make menuselect
```

```
make
```

```
make install
```

```
make samples
```

```
make config
```

```
ldconfig
```


i) Nyní můžeme poprvé spustit proces Asterisk. Ten můžeme spustit několika způsoby:

```
/etc/init.d/asterisk start nebo /etc/init.d/asterisk stop  
asterisk -rvvv
```

B.2 Vytvoření certifikátu pro TLS

Aby bylo možné využívat Secure WebSocket (wss), je třeba vytvořit klíče a certifikáty pro protokol TLS, který zajistí kryptografické zabezpečení přenosu zpráv koncových bodů WebSocket spojení.

a) Vytvoříme si složku, kde budeme vygenerované klíče uchovávat

```
mkdir /etc/asterisk/keys
```

b) Přesuneme se do složky *contrib/scripts*, kde Asterisk poskytuje skript pro vytvoření vlastního, podepsaného certifikátu.

```
cd /usr/src/asterisk-13*  
cd /contrib/scripts
```

c) Spustíme skript, pomocí něhož vygenerujeme vlastní podepsaný certifikát

```
./ast_tls_cert -C <your IP or Domain Name> -O "<name of your company>" -  
d /etc/asterisk/keys
```

Po zavolání skriptu, budete několikrát vyzváni pro zadání hesla.

- Budete vyzváni k zadání hesla pro */etc/asterisk/keys/ca.key*. Tím, dojde k vytvoření */etc/asterisk/keys/ca.crt* file.
- Budete vyzváni, k opětovnému zadání hesla. Tím, dojde k vytvoření */etc/asterisk/keys/asterisk.key*. Soubor */etc/asterisk/keys/asterisk.crt* se vygeneruje automaticky.
- Budete po třetí vyzváni k zadání hesla. Zde dojde k vytvoření souboru */etc/asterisk/keys/asterisk.pem*. Tento soubor je vytvořen kombinací souborů *asterisk.key* a *asterisk.crt*.

Vytvořené certifikáty můžeme zkontrolovat v cílovém adresáři */etc/asterisk/keys* nebo zadáním níže uvedeného příkazu:

```
ls -w 1 /etc/asterisk/keys
```

Měli byste mít vygenerované tyto soubory:

```
asterisk.crt  
asterisk.csr  
asterisk.key  
asterisk.pem  
ca.cfg  
ca.crt  
ca.key  
tmp.cfg
```

Nyní máte nainstalovanou ústřednu PBX Asterisk se všemi predispozicemi pro konfiguraci WebRTC. Konfiguraci WebRTC můžete pokračovat dle kapitoly 10, Konfigurace WebRTC v PBX Asterisk, v hlavním dokumentu této práce.

Příloha C

C.1 Instalace pobočkové ústředny FreeSwitch 1.6

V této příloze je popsán postup instalace pobočkové ústředny FreeSWITCH v LTS verzi 1.6. Dále je zde krok za krokem popsán postup základní konfigurace WebRTC v pobočkové ústředně FreeSWITCH tak, aby na jejím konci byl uživatel schopen provést úspěšný hovor skrze WebRTC klienta za pomoci BPX FreeSWITCH, jakožto signalizačního serveru. Veškerý zde uvedený postup byl proveden na virtualizovaném operačním systému Ubuntu 16.04. Použité příkazy jsou psané takovou formou, aby je bylo možné použít metodou „copy-paste“

C.2 Instalace pobočkové ústředny FreeSWITCH 1.6

- a) Poté co se přihlásíte na váš Ubuntu server je třeba se přepnout na uživatele s právy root.

```
sudo su
```

- b) Dalším krokem je nainstalování všech potřebných balíčků, potřebných pro samotnou kompilaci PBX FreeSWITCH. V tomto kroku také provedeme instalaci většiny programů, které budeme dále potřebovat.

```
apt-get -y install autoconf automake devscripts gawk g++ libjpeg-dev  
libncurses5-dev liblua5.2-dev lua-sec lua-socket git libtool make  
python-dev gawk pkg-config libtiff-dev libperl-dev libgdbm-dev libdb-dev  
gettext libssl-dev libcurl4-openssl-dev libpcre3-dev libspeex-dev  
libspeexdsp-dev libsqlite3-dev libedit-dev libldns-dev libpq-dev  
memcached libmemcached-dev libopus-dev vim tshark curl subversion  
libsndfile-dev
```

- c) Stáhneme si ze stránek yasm.tortall.net/Download.html poslední verzi modulárního assembleru projektu yasm, který posléze vyextrahujeme a nainstalujeme.

```
tar -zxvf yasm-1.3.0.tar.gz
```

```
cd yasm*
```

```
./configure
```

```
make
```

```
make install
```

d) Přesuneme se do složky, kde budeme kompilovat FreeSWITCH.

```
cd /usr/src/
```

e) Stáhneme FreeSWITCH.

```
wget -c http://files.freeswitch.org/freeswitch-1.6.19.tar.gz
```

f) Vyextrahujeme balíček.

```
tar -zxvf freeswitch-1.6.19.tar.gz
```

g) Vstoupíme do složky FreeSWITCH z vyextrahovaného balíčku.

```
cd freeswitch-1.6.19
```

h) Nyní se můžeme pustit do kompilace PBX FreeSWITCH.

```
./configure
```

```
make
```

```
make install
```

```
make cd-sounds-install
```

```
make cd-moh-install
```

i) Nyní můžeme poprvé spustit FreeSWITCH.

```
cd /usr/local/freeswitch/bin
```

```
./freeswitch nebo ./freeswitch -nc
```

C.3 Vytvoření certifikátu pro TLS

FreeSWITCH již při své kompilaci vytvoří dvojici základních certifikátů pro potřebu použití komunikace skrze WebSocket. Jedná se o soubory „*dtls-srtp.pem*“ a „*wss.pem*“, které nalezneme v adresáři */usr/local/freeswitch/certs/*. Tyto soubory jsou zcela dostačující pro základní komunikaci skrze nezabezpečený WebSocket *ws*. Pokud ale budeme chtít provozovat komunikaci skrze zabezpečený WebSocket (*wss*), budeme potřebovat vygenerovat nový, vlastní certifikát.

Protože FreeSWITCH neobsahuje vestavěný skript pro tvorbu vlastního certifikátu, vytvoříme si takovýto skript samy. Jedná se o tentýž skript, který jsme použili v případě PBX Asterisk. Zde jsem provedl pouze malé úpravy jako např. změnu názvů vygenerovaných souborů tak, aby vygenerované soubory odpovídaly pro potřeby PBX FreeSWITCH.

- a) Přesuňte se do adresáře */usr/local/freeswitch/certs*

```
cd /usr/local/freeswitch/certs
```

- b) Smažte původní certifikát *wss.pem*

```
sudo rm -f wss.pem
```

- c) Vytvořte nový soubor, který bude splňovat funkci našeho skriptu pro generování vlastního certifikátu. Já jsem jako název souboru zvolil *gen_ssl.crt*. Název samozřejmě může být libovolný.
- d) Protože text skriptu je opravdu dlouhý na to, abych jej vkládat do textu hlavního souboru, naleznete jej v příloze této práce.

```
Příloha č.2 gen_ssl.crt
```

- e) Nyní spustíme skript pomocí následujícího příkazu:

```
./gen_ssl.crt -C <your IP or Domain Name> -O "<name of your company>" -d  
/usr/local/freeswitch/certs
```

Vytvořené certifikáty můžeme zkontrolovat v cílovém adresáři */usr/local/freeswitch/certs* nebo zadáním níže uvedeného příkazu:

```
ls -w 1 /usr/local/freeswitch/certs
```

Měli byste mít vygenerované tyto soubory:

```
wss.crt  
wss.csr  
wss.key  
wss.pem  
ca.cfg  
ca.crt  
ca.key  
tmp.cfg
```

Nyní máte nainstalovanou ústřednu PBX FreeSWITCH se všemi predispozicemi pro konfiguraci WebRTC. Konfiguraci WebRTC můžete pokračovat dle kapitoly 11, Konfigurace WebRTC v PBX FreeSWITCH, v hlavním dokumentu této práce.

Příloha D

D.1 Laboratorní úloha 6 - PBX Asterisk – WebRTC

6.1 Úvod

V minulém cvičení jste se seznámili se základní konfigurací koncových bodů (účtů), využívající modernější kanálový ovladač PJSIP. V tomto cvičení se seznámíte s konfigurací integrovaného HTTP serveru, pro podporu transportní metody přes zabezpečený WebSocket. Dále rozšíříte původní konfiguraci koncových bodů PJSIP tak, aby byly schopny využívat WebRTC komunikátory, díky nimž zprostředkujete videohovor přímo z prostředí webového prohlížeče.

6.2 WebRTC v PBX Asterisk

WebRTC je Open-Source projekt propagovaný společností Google, který umožňuje komunikaci v reálném čase přímo v prostředí prohlížečů, bez nutnosti instalace rozšiřujících zásuvných modulů. PBX Asterisk podporuje WebRTC již od verze 11, kdy bylo pro potřeby vývojářů k navázání komunikace mezi ústřednou a WebRTC klienty, vytvořen modul `res_http_websocket`. Aby bylo možné použít SIP jako signalizační protokol, byla podpora WebSocketu přidána do kanálového ovladače `chan_sip`, později do `chan_pjsip`. Asterisk ve verzi 11 také implementoval podporu podpůrných protokolů jako ICE, STUN a TURN do `res_rtp_asterisk`, které dovolují klientům za síťovým překladačem NAT lépe komunikovat s ústřednou. Asterisk od verze 11 podporu pro WebRTC klienty nadále rozšiřuje, neboť lze předpokládat, že počet WebRTC klientů bude do budoucna narůstat.

Tato úloha demonstruje základní podporu a funkčnost WebRTC v PBX Asterisk. Asterisk bude nakonfigurován tak, aby podporoval vzdáleného WebRTC klienta Sipml5. Pomocí webového prohlížeče Firefox ve verzi 54.0 a vyšší si zaregistrujete Sipml5 klienta k ústředně a provedete videohovor.

6.2.1 Konfigurace – základní predispozice

Aby byl Sipml5 klient schopen komunikace s PBX Asterisk, musí být ústředna nainstalována s podporou těchto kanálových ovladačů:

- `res_crypto` – zajišťuje zdroje pro kryptografické aplikace
- `res_http_websocket` – podpora transportní metody WebSocket pro `chan_sip`
- `res_pjsip_transport_websocket` – podpora transportní metody WebSocket pro `chan_pjsip`

Chcete-li zkontrolovat, zda máte tyto moduly dostupné, můžete použít následující příkaz:

```
# ls -w 1 /usr/lib/asterisk/modules/{*crypto*,*websocket*}
```

Měli byste dostat tento výstup:

```
/usr/lib/asterisk/modules/res_crypto.so  
/usr/lib/asterisk/modules/res_http_websocket.so  
/usr/lib/asterisk/modules/res_pjsip_transport_websocket.so
```

Dále zkontrolujte, zda má Asterisk tyto moduly načtené:

```
# asterisk -rx "module show like crypto"  
# asterisk -rx "module show like websocket"
```

Měli byste dostat tento výstup:

```
# asterisk -rx "module show like crypto"  
Module                Description                Use Count    Status  
res_crypto.so         Cryptographic Digital Signatures    1            Running  
1 modules loaded  
  
# asterisk -rx "module show like websocket"  
Module                Description                Use Count    Status  
res_http_websocket.so HTTP WebSocket Support          3            Running  
res_pjsip_transport_websocket.so PJSIP WebSocket Transport Support 0            Running  
2 modules loaded
```

Pokud nemáte moduly načteny, zkontrolujte konfigurační soubor `/etc/asterisk/modules.conf` a ujistěte se, že nemáte tyto moduly explicitně vypnuty.

6.2.2 Konfigurace – certifikáty

Moderní prohlížeče vyžadují použití protokolů TLS pro šifrování signalizace a DTLS-SRTP pro šifrování médií. Aby Asterisk mohl používat protokoly TLS a DTLS-SRTP, je nutné načíst certifikáty, pomoci nichž, získá Asterisk potřebné klíče pro šifrování komunikace.

Je možné použít vlastně podepsané certifikáty, Asterisk poskytuje přímo skript k jejich vytvoření ve zdrojovém adresáři /contrib/scripts. Nicméně mnohé prohlížeče vyžadují použití veřejně podepsaných certifikátů od známých certifikačních autorit. V rámci této úlohy jsou certifikáty již předpřipravené a podepsané místní, známou certifikační autoritou. Certifikáty samotné můžete nalézt v adresáři /etc/asterisk_keys.

6.2.3 Konfigurace – HTTP serveru

WebRTC klient Sipml5 používá WebSocket jako transportní metodu pro výměnu signalizačních zpráv. Pro zprovoznění komunikace mezi Asterisk serverem a Sipml5 klientem je třeba na straně Asterisk serveru vytvořit HTTP server, ke kterému se bude klient připojovat a sestavovat WebSocket spojení. Asterisk pro tento případ obsahuje vlastní integrovaný HTTP server. Pro konfiguraci tohoto HTTP serveru slouží konfigurační soubor http.conf. Níže je uvedený příklad konfigurace HTTP serveru.

```
[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlscipher=AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA
tlsservercipherorder=yes
tlsbindaddr=0.0.0.0:8089
tlscertfile=/etc/asterisk/keys/asterisk.pem
```

- enable – tento parametr určuje, zda je či není HTTP/HTTPS povoleno. Výchozí hodnota je „no“.
- bindaddr – představuje adresu na kterou se váže HTTP protokol. V tomto případě je ve výchozím nastavení použita obecná adresa.
- bindport – zde se definuje port na kterém bude naslouchat HTTP server. Výchozí hodnota je 8088.
- tlsenable – tento parametr určuje, zda bude povoleno použití protokolu TLS pro zabezpečení komunikace.
- tlsservercipherorder – zde můžete určit, které kryptografické šifry bude HTTPS server podporovat, resp. nabízet ve svých „Server Hello“ zprávách, při navazování zabezpečeného spojení.
- tlsbindaddr – představuje adresu na kterou se váže HTTPS protokol. V tomto případě je ve výchozím nastavení použita obecná adresa.
- tlsbindport – zde se definuje port na kterém bude naslouchat HTTPS server. Výchozí hodnota je 8089.
- tlscertfile – tento parametr slouží pro definování cesty k certifikačnímu souboru, který obsahuje především klíč, kterým bude server budoucí komunikaci šifrovat.

6.2.4 Konfigurace – PJSIP

Dále bude třeba nakonfigurovat účty využívající stack PJSIP. Tyto účty se konfigurují v souboru pjsip.conf a jsou rozděleny do několika sekcí, s kterými jste se seznámili blíže v Lab5b. Tyto účty dále rozšíříme o nastavení nezbytné pro podporu WebRTC klientů.

WebRTC klient Sipml5 využívá WebSocket jako transportní metodu pro připojení k HTTP/HTTPS serveru PBX Asterisk. Stejně tak, jako byl nakonfigurován HTTP/HTTPS server, aby naslouchal WebSocket přenosům, musí být PJSIP nakonfigurován pro podporu transportní metody WebSocket. V níže uvedeném příkladu je nakonfigurován PJSIP s podporou pro zabezpečený WebSocket („Secure WebSocket“ – wss). Veškeré úpravy jsou provedeny v konfiguračním souboru pjsip.conf.

```

[transport_wss]                ;název sekce
type=transport                 ;typ sekce
protocol=wss                   ;protokol
bind=0.0.0.0:5067              ;adresa a port, kde PJSIP naslouchá

[1000]                         ;název sekce
type=aor                       ;typ sekce
max_contacts=1                ;maximální počet registrovaných zařízení
remove_existing=yes           ;povolí úspěšnou registraci zařízení nad
                               ;počet maximálního počtu registrací,
                               ;odhlášením zařízení s nejstarší registrací

[1000]                         ;název sekce
type=auth                     ;typ sekce
auth_type=userpass            ;typ ověřování
username=1000                 ;uživatelské jméno
password=asdf                 ;uživatelské heslo

[1000]                         ;název sekce
type=endpoint                 ;typ sekce
transport=transport_wss       ;přiřazení transportní metody
aors=1000                     ;přiřazení sekce AOR
auth=1000                     ;přiřazení sekce AUTH
use_avpf=yes                  ;použití zpětnovazebních zpráv pro rtcp
media_encryption=dtls         ;určuje typ šifrování médií
dtls_ca_file=/etc/asterisk_keys/ca.pem ;cesta k certifikátu certifikační autority
dtls_cert_file=/etc/asterisk_keys/asterisk.pem ;cesta k certifikátu, který bude použit
dtls_verify=fingerprint       ;typ ověřování
dtls_setup=actpass            ;typ akceptovaného připojení pro dtls
ice_support=yes               ;podpora podpůrného protokolu ICE
context=kontext1              ;přiřazení účtu do kontextu
disallow=all                  ;zakázání všech podporovaných kodeků
allow=alaw, vp8               ;explicitní povolení kodeků alaw a vp8

```

6.3 Zadání úlohy – konfigurace WebRTC

📖 Úkol 1: Seznamte se se způsobem konfigurace HTTP serveru dle kapitoly 6.2.3.

🔧 Úkol 2: Nakonfigurujte HTTP server pro WebSocket spojení

Podle vzoru níže, nakonfigurujte vestavěný HTTP server v souboru `http.conf`.

```

[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlscipher=AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA
tlsservercipherorder=yes
tlsbindaddr=0.0.0.0:8089
tlscertfile=/etc/asterisk_keys/asterisk.pem

```

Jako IP adresu můžete použít IP adresu PC vašeho pracoviště dle Tab. 1, případně ponechat defaultní adresu 0.0.0.0, která vám zpřístupní HTTP server na adrese libovolného eth rozhraní Asterisk serveru. Definujte hodnotu podporovaných šifer na „*AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA*“ a specifikujte konkrétní

cestu k certifikačnímu souboru Asterisku, který obsahuje klíč pro šifrování komunikace. Reloadujte nastavení HTTP serveru v konzoli Asterisku příkazem *reload http*. Ověřte správnost konfigurace a porty na kterých HTTP server naslouchá příkazem *http show status*.

```
HTTP Server Status:  
Prefix:  
Server: Asterisk/certified/13.13-cert4  
Server Enabled and Bound to 0.0.0.0:8088  
  
HTTPS Server Enabled and Bound to 0.0.0.0:8089
```

Obr.1 Výpis konfigurace HTTP serveru

Přístup na HTTP/HTTPS server můžete ověřit i přímo v prohlížeči, pokud zadáte do URL adresu `https://<IP_adresa_vášeho_pracoviště>:8089`. V tomto případě však prohlížeč bude vyžadovat přiřazení výjimky, neboť certifikát `asterisk.pem`, podepsaný lokální, známou certifikační autoritou, na který se odkazujeme v nastavení HTTP serveru, obsahuje záznam pouze pro veřejný doménový název Asterisk serveru. Zadejte tedy do URL adresu `https://pc2XX.lab5-34.utko.feec.vutbr.cz:8089`, kde XX představuje číslo vašeho pracoviště. Cílová URL adresa například pro pracoviště 1 tedy bude vypadat následovně: `https://pc201.lab5-34.utko.feec.vutbr.cz:8089`. Měla by se Vám načíst níže uvedená stránka.

Not Found

The requested URL was not found on this server.

Asterisk/certified/13.13-cert4

Obr.2 Veřejná adresa HTTPS serveru

✂ Úkol 3: Rozšiřte vhodně konfiguraci PJSIP stacku dle kapitoly 6.2.4, aby koncové účty podporovaly připojení k WebRTC klientu Sipml5

Podle stávajícího koncového účtu 500 v `pjsip.conf` vytvořte další účty (sekce `ENDPOINT`) s názvy 1000 a 2000, a k nim potřebné sekce (`AUTH`, `AOR`). Sekce typu `AOR` musí mít shodný název se sekci `ENDPOINT`, ke které je přiřazena. Sekce `AUTH` může mít libovolný název, v sekci `ENDPOINT` však musíme tuto sekci správně přiřadit. Tyto účty rozšiřte dle vzoru níže vhodným způsobem tak, aby byly schopny přihlášení přes WebRTC klienta Sipml5. Po dokončení úprav proveďte reload PJSIP stacku příkazem `pjsip reload` a ověřte správnost konfigurace příkazem `pjsip show endpoints`. Tento příkaz vám vypíše dostupné PJSIP účty.

```
[1000]
type=aor
max_contacts=1
remove_existing=yes

[1000]
type=auth
auth_type=userpass
username=1000
password=asdf

[1000]
type=endpoint
transport=transport_wss
aors=1000
auth=1000
use_avpf=yes
media_encryption=dtls
dtls_ca_file=/etc/asterisk_keys/ca.pem
dtls_cert_file=/etc/asterisk_keys/asterisk.pem
dtls_verify=fingerprint
dtls_setup=actpass
ice_support=yes
context=kontext1
disallow=all
allow=alaw, vp8
```

Protože WebRTC klient Sipml5 využívá transportní metodu `WebSocket`, je třeba tuto transportní metodu v PJSIP stacku definovat. Přidejte tedy na začátek konfiguračního souboru `pjsip.conf` definici této transportní metody, dle ukázky níže.

```
[transport_wss]
type=transport
protocol=wss
bind=0.0.0.0:5067
```

✂ Úkol 4: Upravte směrovací pravidla, aby Asterisk využíval pro účty 500, 1000 a 2000 PJSIP stack

Upravte směrovací pravidla v patřičných kontextech tak, aby hovory na klapky 500, 1000 a 2000 využívaly stack PJSIP. K volání klapky PJSIP stackem místo nativním SIP stackem, lze dosáhnout obdobným způsobem, který je uveden níže. Nezapomeňte poté reloadovat modul pro směrování příkazem *dialplan reload*.

```
[kontext1]
exten =>500,1,Dial(PJSIP/500)
```

✂ Úkol 5: Zaregistrujte WebRTC klienta Sipml5 na jeden z vytvořených účtů, pomocí desktopového internetového prohlížeče

WebRTC klient Sipml5 podporuje širokou škálu desktopových internetových prohlížečů od Google Chrome, Firefox, Operu, Safari až po Internet Explorer a další. Otevřete internetový prohlížeč a přejděte na adresu <https://pc2XX.lab5-34.utko.feec.vutbr.cz/sipml5>, kde XX představuje číslo vašeho pracoviště. Cílová URL adresa například pro pracoviště 1 tedy bude vypadat následovně: <https://pc201.lab5-34.utko.feec.vutbr.cz/sipml5>

Vyplňte registrační box Sipml5 klienta dle přiloženého Obr. 3.

Registration

Display Name:

Private Identity*:

Public Identity*:

Password:

Realm*:

* Mandatory Field

Obr.3 Registrační box Sipml5 klienta

Příklad pro uživatele 1000 na pracovišti 1 bude vypadat následovně:

Display Name: Franta
Private Identity: 1000
Public Identity: sip:1000@ pc201.lab5-34.utko.feec.vutbr.cz
Password: asdf
Realm: pc201.lab5-34.utko.feec.vutbr.cz

Pokračujte v konfiguraci Sipml5 klienta přechodem do jeho rozšířeného nastavení kliknutím na tlačítko „Expert mode?“. Zde v tomto nastavení bude třeba definovat na jaké adrese a portu naslouchá vámi dříve nakonfigurovaný HTTPS server, resp. kde očekává WebSocket spojení. Vyplňte rozšířené nastavení dle níže přiloženého Obr. 4.

Saved

Expert settings

Disable Video:

Enable RTCWeb Breaker^[1]:

WebSocket Server URL^[2]:

SIP outbound Proxy URL^[3]:

ICE Servers^[4]:

Max bandwidth (kbps)^[5]:

Video size^[6]:

Disable 3GPP Early IMS^[7]:

Disable debug messages^[8]:

Cache the media stream^[9]:

Disable Call button options^[10]:

Save Revert

Obr. 4 Rozšířené nastavení

V nově otevřeném expert settings boxu, do pole WebSocket Server URL zadejte: wss://pc2XX.lab5-34.utko.feec.vutbr.cz:8089/ws. XX zde představuje číslo vašeho pracoviště. Pro pracoviště 1 bude tedy WebSocket Server URL obsahovat adresu wss://pc201.lab5-34.utko.feec.vutbr.cz:8089/ws

Dále definujte adresu STUN/TURN serveru v poli ICE Servers. Adresa STUN/TURN serveru je pro všechna pracoviště stejná. [{url: 'stun:turn.lab5-34.utko.feec.vutbr.cz:3478'}].

Nastavení uložte a vraťte se zpět na hlavní stránku s registračním boxem. Přihlaste se k ústředně a zkontrolujte správnost přihlášení pomocí příkazu *pjsip show endpoints*.

```
Endpoint: 1000                                Not in use    0 of inf
  InAuth: 1000/1000
    Aor: 1000                                  1
  Contact: 1000/sips:1000@192.168.1.19:59852;transpor 216c72dd69 Unknown    nan
Transport: transport-wss                      wss          0          0 0.0.0.0:5060
```





Obr. 5 Registrace PJSIP Endpoint

Zaregistrujte si PJSIP účet 500 k VoIP telefonu Planet VIP-2020PT obdobným způsobem, kterým jste zaregistrovali standardní SIP účet v úkolu č.2 v Lab1. Proveďte testovací hovor mezi Sipml5 klientem z internetového prohlížeče a VoIP telefonem Planet VIP-2020PT. Hovor ze strany Sipml5 klienta provedete rozbalením nabídky Call a vybráním možnosti „Audio“. Jelikož PC nedisponuje zvukovým výstupem ani mikrofonom, hovor bude hluchý, nicméně byste se měli úspěšně dovolat oběma směry. Průběh hovoru můžete sledovat z konzole Asterisků. Případně výpis konzole rozšířte o debug SIP komunikace zadáním příkazu *sip set debug on*.

✂ Úkol 6: Zaregistrujte WebRTC klienta Sipml5 na jeden z vytvořených účtů, pomocí mobilního internetového prohlížeče

U mobilních internetových prohlížečů je zatím situace značně omezená a doporučujeme využívat Firefox ve verzi 59.0.2 a vyšší. Stáhněte a nainstalujte „Firefox – rychlý prohlížeč“ do svého smartphonu z oficiálních zdrojů (např. Obchod play nebo www.apkmirror.com). Po úspěšné instalaci, otevřete prohlížeč „Firefox – rychlý prohlížeč“ a přejděte na stránku <https://pc2XX.lab5-34.utko.feec.vutbr.cz/index>, kde XX představuje číslo vašeho pracoviště. Cílová adresa například pro pracoviště 1 tedy bude vypadat následovně: <https://pc201.lab5-34.utko.feec.vutbr.cz/sipml5>. Stáhněte si a nainstalujte certifikát lokální, známe certifikační autority, 534.pem. Při instalaci certifikátu zvolte možnost „Instalovat pro VPN a aplikace“. Po úspěšném nahrání certifikátu do mobilního prohlížeče, přejděte na adresu <https://pc2XX.lab5-34.utko.feec.vutbr.cz/sipml5>, kde XX představuje číslo vašeho pracoviště. Cílová URL adresa například pro pracoviště 1 tedy bude vypadat následovně: <https://pc201.lab5-34.utko.feec.vutbr.cz/sipml5>. Zaregistrujte se obdobně jako v předchozím

úkolu č. 5. Proved'te testovací hovor mezi mobilním WebRTC klientem a VoIP telefonem Planet VIP-2020PT.

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 534.pem	03-May-2018 16:39	1.5K	
 sipml5/	08-Feb-2018 18:45	-	
 voiceone/	10-Oct-2011 16:00	-	
 voiceone_webservices/	10-Oct-2011 17:40	-	

Obr. 6 Stažení certifikátu

✂ Úkol 7: Vytvořte SIP trunk k nadřazené ústředně Anča (192.168.10.5) s využitím PJSIP stacku

Nejprve zkontrolujte, zda není v souboru sip.conf vytvořený trunk účet na nadřazenou ústřednu a popřípadě jej odstraňte a relaodujte sip stack příkazem *sip reload*. Vytvořte SIP trunk v PJSIP stacku mezi vaší ústřednou a nadřazenou ústřednou Anča (192.168.10.5). Konfigurace trunku pro PJSIP stack se provádí v konfiguračním souboru pjsip_wizard.conf. V tomto souboru již naleznete předkonfigurovaný trunk. Předlohu odkomentuje a upravte tak, aby registrační údaje odpovídaly trunku, přiřazeného k vašemu pracovišti dle Tab.1. Níže je uveden příklad konfigurace trunku SIP_TRN_1 pro pracoviště 2.

```
[SIP_TRN_1] ;navez
type=wizard ;typ sekce
sends_auth=yes ;overovani
sends_registrations=yes ;odeslat registraci
remote_hosts=192.168.10.5 ;adresa poskytovatele
outbound_auth/username=SIP_TRN_1 ;uzivatelske jmeno
outbound_auth/password=anca ;heslo
endpoint/context=prichozi_kontext ;kontext
endpoint/allow=alaw ;povoleni kodeku alaw
endpoint/allow=vp8 ;povoleni kodeku vp8
```

6.3.1 Tab. 1: Rozdělení pracovišť

Pracoviště	IP adresa PC	IP adresa telefonu	klapky	Jméno trunku – IP 192.168.10.5
1	192.168.10.101	192.168.10.161	60X	SIP_TRN_0 nebo IAX2_TRN_0
2	192.168.10.102	192.168.10.162	61X	SIP_TRN_1 nebo IAX2_TRN_1
3	192.168.10.103	192.168.10.163	62X	SIP_TRN_2 nebo IAX2_TRN_2
4	192.168.10.104	192.168.10.164	63X	SIP_TRN_3 nebo IAX2_TRN_3
5	192.168.10.105	192.168.10.165	64X	SIP_TRN_4 nebo IAX2_TRN_4
6	192.168.10.106	192.168.10.166	65X	SIP_TRN_5 nebo IAX2_TRN_5
7	192.168.10.107	192.168.10.167	66X	SIP_TRN_6 nebo IAX2_TRN_6
8	192.168.10.108	192.168.10.168	67X	SIP_TRN_7 nebo IAX2_TRN_7
9	192.168.10.109	192.168.10.169	68X	SIP_TRN_8 nebo IAX2_TRN_8
10	192.168.10.110	192.168.10.170	69X	SIP_TRN_9 nebo IAX2_TRN_9

Tab.1: Rozdělení pracovišť

Pozn.: Heslo je pro všechny trunky **anca**.

Opět restartujte PJSIP stack příkazem *pjsip reload* a ověřte úspěšnou registraci k nadřazené ústředně příkazem *pjsip show registrations*. Na Obr.7 je výpis úspěšně registrovaného trunku SIP_TRN_1 pro pracoviště 2.

```
<Registration/ServerURI.....> <Auth.....> <Status.....>
=====
SIP_TRN_1-reg-0/sip:192.168.10.5          SIP_TRN_1-oauth  Registered
```

Obr.7 Registrace trunku k ústředně

✂ Úkol 8: Upravte směrovací plán tak, aby se bylo možno dovolat prostřednictvím SIP trunku na ostatní pracoviště.

Upravte vhodně směrovací pravidla v souboru *extensions.conf*, dle Lab5b, úkol 6 a 7 tak, aby se bylo možno dovolat do vnější sítě a aby byla příchozí volání z vnější sítě směrována na interní klapky. Ověřte funkčnost realizací hovorů mezi WebRTC klienty, přihlášenými k ústřednám na různých pracovištích.

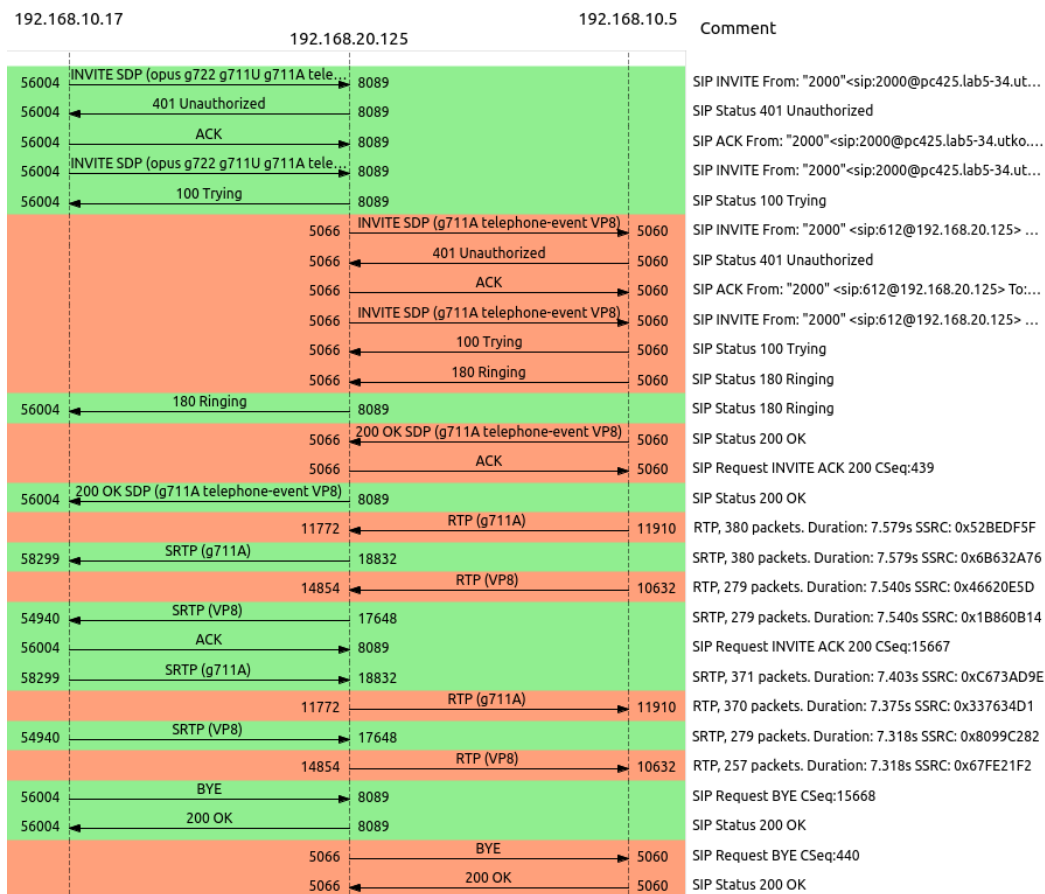
✂ Úkol 9: Zachycení a analyzování hovoru mezi WebRTC klienty.

Pomocí paketového analyzátoru Wireshark zachyťte a analyzujte hovor realizovaný pomocí WebRTC klienta Sipml5. Protože je komunikace šifrovaná, je třeba do programu Wireshark nahrát certifikát, který obsahuje klíč pro dešifrování. Spusťte program Wireshark a v horním panelu přejděte do Edit->Preferences..->Protocols. Zde vyhledejte protokol SSL. V nastavení protokolu SSL u položky „RSA keys list“ klikněte na „Edit...“ a definujte klíč dle Obr. 8.

IP address	Port	Protocol	Key File	Password
any	8089	http	/etc/asterisk_keys/asterisk.pem	

Obr. 8 Definování klíče pro dešifrování

Nastavení uložte a spusťte nahrávání na rozhraní Eth0. Pro lepší orientaci v paketovém toku, použijte filter *sip*. Pokud zachycená SIP komunikace bude stále zašifrovaná v těle TCP segmentů, proveďte restart procesu asterisk pomocí příkazu *core restart now*. Pro správné aplikování klíče pro dešifrování je nutné, aby program Wireshark zachytil navázání komunikace. Zachycenou komunikaci si můžete nechat detailně zobrazit v nabídce Telephony->VoIP Calls->Flow Sequence. Na níže uvedeném Obr. 9 je zachycena komunikace WebRTC klienta, registrovaného z IP 192.168.10.17 k účtu 2000 na Asterisk ústředně s IP 192.168.20.125. Tento klient inicializuje hovor na klapku 612 veřejného číslovacího plánu, proto jsou zprávy přeposílány skrze trunk, nadřazené ústředně Anča s IP 192.168.10.5.



obr. 9 Ukázka zachycené SIP komunikace

Seznam použité literatury

[1] Asterisk 12 Part IV: *The SIP Stack of the Future*: Matt Jordan. Blogs.digium.com [online]. [cit. 2016-11-8]. Dostupné z: <http://blogs.digium.com/2013/11/20/asterisk-12-part-iv-sip-stack-future/>

[2] Asterisk Project: WebRTC tutorial [online]. 30. 4. 2018 [cit. 2018-05-15]. Dostupné z: <https://wiki.asterisk.org/wiki/display/AST/WebRTC+tutorial+using+SIPML5>

[3] Asterisk Versions. Asterisk.org [online]. Russell Bryant, 2016 [cit. 2016-10-20].

Dostupné z: <https://wiki.asterisk.org/wiki/display/AST/Asterisk+Versions>

[4] ŠALKO Jaroslav, Bc. *Implementace WebRTC v Open source PBX*. Brno, 2018. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií Ústav telekomunikací Technická 12, 616 00 Brno.

[5] ŠILHAVÝ, PH.D, Ing. Pavel. *Telekomunikační a informační systémy*. BRNO, 2014. Vysoké učení technické v Brně Fakulta elektrotechniky a komunikačních technologií Ústav telekomunikací Technická 12, 616 00 Brno.

Příloha E

E.1 CD/DVD

Tato příloha obsahuje konfigurační soubory a zachycené nahrávky hovorů vybraných PBX, v elektronické podobě, nacházející se na přiloženém CD/DVD. Na CD/DVD se konkrétně nacházejí položky uvedené v seznamu níže.

- E.1 PBX_Asterisk
 - E.11 – Obsahuje konfigurační soubory *extensions.conf*, *http.conf*, *pjsip.conf* a *sip.conf*, již nakonfigurované ústředny podporující WebRTC.
 - E.12 – Obsahuje nahrávku *asterisk.pcapng*, zachyceného videohovoru mezi WebRTC klienty, a privátní klíč *asterisk.key* pro dešifrování komunikace

- E.2 PBX_FreeSWITCH
 - E.21 – Obsahuje konfigurační soubory *1000.xml*, *1001.xml*, *default.xml*, *internal.xml* a *vars.xml*, již nakonfigurované ústředny podporující WebRTC
 - E.22 - Obsahuje nahrávku *freeswitch.pcapng*, zachyceného videohovoru mezi WebRTC klienty, a privátní klíč *wss.key* pro dešifrování komunikace
 - E.23 – Obsahuje script *gen_ssl_crt*, pro vygenerování vlastně podepsaných certifikátů pro PBX FreeSWITCH

- E.3 Laboratorní úloha
 - E.31 - Obsahuje konfigurační soubory *extensions.conf*, *http.conf*, *pjsip.conf*, *sip.conf* a *wizard.conf*, již nakonfigurované ústředny podporující WebRTC.

Seznam použitých zkratk

API	Application Programming Interface
DH	Diffie-Hellman
DTLS	Datagram Transport Layer Security
ECDH	Elliptic-Curve Diffie – Hellman
GSM	Global System for Mobile Communications
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
ISO/OS	International Standards Organization Open Systems Interconnection
LTE	Long Term Evolution
LTS	Long Term Support
NAT	Network Address Translation
P2P	Peer to Peer
PBX	Private branch exchange
PC	Personal Computer
PHP	Hypertext Preprocessor
RFC	Request for Comments
RTP	Real-Time Protocol
SCTP	Stream Control Transmission Protocol
SIP	Session Initiation Protocol
SFU	Selective Forwarding Unit
SRTP	Secure Real-Time Transport Protocol
SS7	Signaling System Number 7
SSL	Secure Sockets Layer
STUN	Session Traversal Utilities for Nat

TCP	Transmission Control Protocol
TLS	Transport Layer Security
TURN	Traversal Using Relay NAT
UA	User Agent
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VoIP	Voice over Internet Protocol
W3C	World Wide Web Consortium
WebRTC	Web Real-Time Communications
WS	WebSocket
WSS	Secure WebSocket
XHR	XML Http Request
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Seznam tabulek

Tabulka 1 Prohlížeče podporující WebRTC.....	15
Tabulka 2 PBX Asterisk - podpora WebRTC.....	29
Tabulka 3 PBX FreeSWITCH - podpora WebRTC.....	31
Tabulka 4 PBX Kamilio - podpora WebRTC	33

Seznam obrázků

Obrázek 1 NAT - demonstrační ilustrace	21
Obrázek 2 STUN - demonstrační ilustrace	22
Obrázek 3 WebRTC - schéma navazování spojení 1	23
Obrázek 4 WebRTC - signalizace v ideálních podmínkách	24
Obrázek 5 WebRTC - signalizace v reálných podmínkách	25
Obrázek 6 WebRTC - schéma navazování spojení 2	26
Obrázek 7 WebRTC - signalizace v reálných podmínkách (STUN)	26
Obrázek 8 WebRTC - signalizace v reálných podmínkách (TURN)	27
Obrázek 9 Schéma zapojení.....	34
Obrázek 10 Sipml5 klient - okno registrace	36
Obrázek 11 Mozilla Firefox - import certifikátu 2	42
Obrázek 12 Wireshark - šifrovaná SIP komunikace	46
Obrázek 13 Wireshark - ukázka TCP paketu.....	46
Obrázek 14 Wireshark - import klíče pro dešifrování 1	47
Obrázek 15 Wireshark - import klíče pro dešifrování 2	47
Obrázek 16 Wireshark - dešifrovaná SIP komunikace.....	48
Obrázek 17 Wireshark - tcp hand-shake, zpráva server hello	49
Obrázek 18 Wireshark - http-over-tls, http get-upgrade.....	49
Obrázek 19 Wireshark - SIP flow 1	50
Obrázek 20 Wireshark - SIP flow 2	56