МКБ

Организация IP-телефонии в iOS мобильном приложении.

май 2022 г.

Иванов Олег Олегович

МКБ

Иванов Олег

Руководитель направления центра компетенций дистанционных каналов обслуживания ИТ-дирекции.

Выпускник Тверского госуниверситета. В студенчестве работал на полставки в НИИИТ, программируя на C++, Qt. Разрабатывал как серверные, так и клиентские части программных продуктов. Занимался шифрованием данных, разрабатывая криптоконтейнеры. Последнее время занимаюсь разработкой системы мобильного банкинга.

К команде Московского кредитного банка присоединился в 2013 году.

Цель и план

Цель:

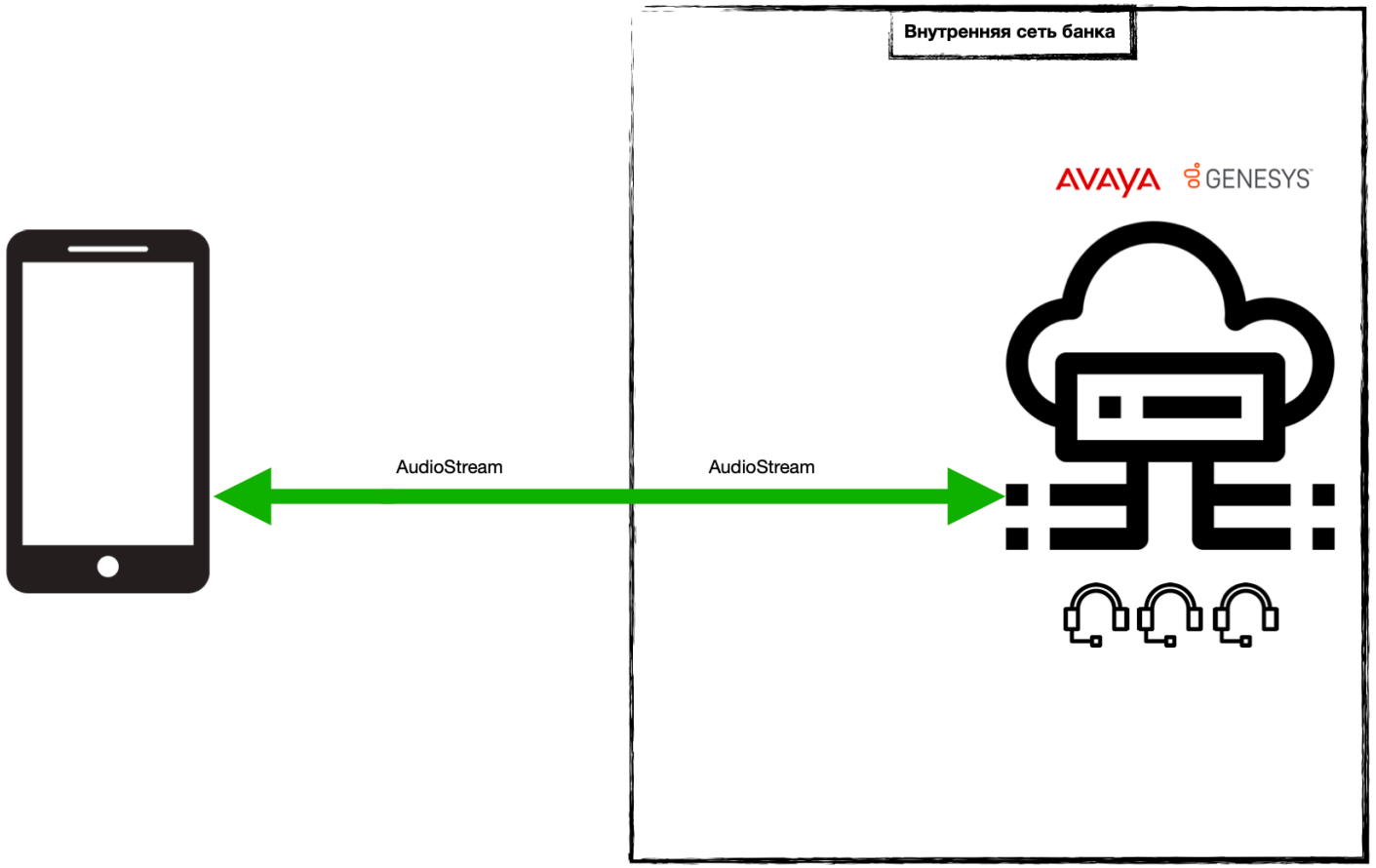
Необходимо организовать звонок по ‘Интернету’ в Call Center во внутреннюю АТС через iOS мобильное приложение(см. схему на 4 слайде ниже)

План:

- Понять, что такое звонок по ‘Интернету’ в разрезе теории
- Рассмотреть протоколы, которые реализуют организацию звонка
- Рассмотреть конкретные решения, реализующие эти протоколы
- Все выводы доклада будем отображать на схеме 4 слайда

P.S. Все решения показанные в докладе применимы и для реализации в Android мобильном приложении.

Схема организации звонка по ‘Интернету’



IP-телефония.

IP-телефония - это голосовая связь, видеообщение и текстовые сообщения, прием и передача которых осуществляется посредством протокола IP по сетям передачи данных.

VoIP — «Voice over IP», иначе «голос через интернет-соединение»

Это система передачи голосовых сообщений посредством интернет протоколов.

VoIP позволяет вести телефонные переговоры как через глобальную сеть Internet, так и по локальной сети.

VoIP Преимущества/Недостатки

Преимущества:

- организация конференцсвязи;
- передача данных (фото, видео) параллельная с разговором;
- низкая стоимость междугородних и международных разговоров;
- отсутствие затрат на переговоры по внутрикорпоративной сети.

Недостатки:

- прямая зависимость качества звука от пропускной способности Internet канала;
- возможна задержка голосового сигнала;
- безопасность и конфиденциальность разговоров по VoIP зависит от качества применяемого оборудования;
- невозможность проведения переговоров при отключении электроэнергии.

SIP-телефония

На основании стандарта VOIP выросла SIP-телефония.

SIP-телефония — технология на базе SIP (Session Initiation Protocol) протокола — это протокол для передачи мультимедийной информации. для совершения телефонных звонков.

SIP-телефония имеет более широкие возможности. Он предоставляет возможность не только голосового общения, но также трансляции мультимедийной информации, обмена текстовыми сообщениями и т.д.

Итог:

VoIP — общая технология передачи голоса через интернет, которая используется для разных задач.

SIP-телефония — одна из разновидностей IP-телефонии, в которой используется VoIP.

Протоколы для организации звонка

1. **Сигнальные протоколы**.

   Они предназначены для координации участников взаимодействия, они используются для установки и завершения звонков, согласования параметров связи и других задач.

2. **Протоколы передачи медиа данных.**

   Переда медиа потоков(данных).

Сигнальные протоколы

- Протокол H.323
- Skinny (SCCP)
- H.248(MEGACO)
- IAX2 (Inter-Asterisk eXchange protocol)
- SIP (Session Initiation Protocol)

Протоколы передачи медиа данных

- RTP (Real-time Transport Protocol)
- SRTP (Secure Real-time Transport Protocol)

SIP протокол

SIP протокол - это «протокол инициирования сеансов», поэтому он отвечает за установление соединения.

Он находит абонента и обеспечивает стабильность связи, но не передает непосредственно текст, голос или видео.

SIP является **текстовым протоколом** и его синтаксис во многом схож с HTTP.

Сообщения в SIP разделяются на **запросы и ответы**.

Основные SIP запросы

REGISTER — регистрация URI клиента на сервере регистрации;

INVITE — инициализация протокола для установления сеанса;

ACK — подтверждение инициализации сеанса со стороны клиента;

BYE — завершение сеанса;

CANCEL — отмена любого ожидающего запроса;

UPDATE — изменение состояния сеанса без изменения диалога;

REFER — запрос на оформление переадресации;

INFO — информация о сеансе, без его модификации;

OPTIONS — запрос информации о функциональности сервера.

SIP ответы

1XX — коды предварительного состояния сеанса. Указывают на то, что запрос был получен сервером и поступил в обработку;

2XX — успешное завершение запроса;

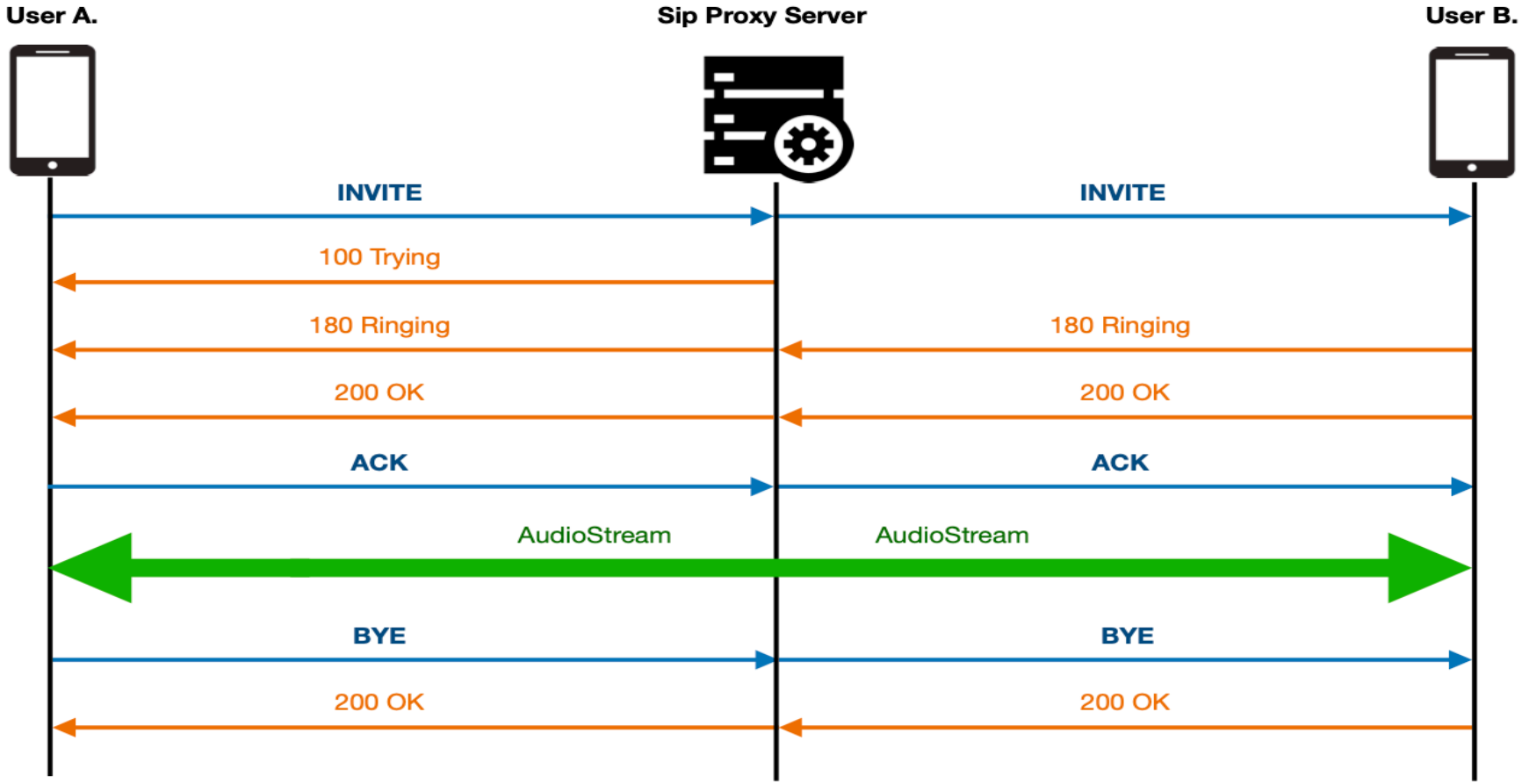
3XX — перенаправление запроса с указанием необходимости заполнения его новым адресом назначения;

4XX — отклонение запроса, либо указание на ошибку в запросе;

5XX — отказ в выполнении запроса;

6XX — отклонение запроса по причине невозможности установить соединение.

Схема организации звонка с помощью SIP



SIP запросы

Для того чтобы вскрыть топологию SIP запросов мы использовали приложение **Linphone**. Плюс использовали утилиту **tcpdump**. Также неплохо использовать **Wireshark** — программу-анализатор трафика.

```
CANCEL sip:xxxx@1.1.1.1 SIP/2.0
Via: SIP/2.0/UDP 1.1.1.1:52384;branch=z9hG4bK.INyvA3CW1;rport
Call-ID: bRhQJKvXIY
From: <sip:yyyy@1.1.1.2>;tag=5L2in8Rvi
To: "xxxx" <sip:xxxx@1.1.1.2>
Max-Forwards: 70
CSeq: 20 CANCEL
User-Agent: Linphone_simulator.64bits_iOS10.2/3.15-93-gfeda86f (belle-sip/1.5.0)
```

Основные параметры SIP запросов

Поля **Via, Max-Forwards, To, From, Call-ID и CSeq** составляют минимальный необходимый набор полей заголовков SIP-запросов.

Via — Путь, пройденный запросом на данный момент.
To — Указывает получателя.
Call-ID — Глобальный уникальный идентификатор для вызова.
From — От кого поступает вызов.
Cseq — Однозначно определяет транзакции в диалоге.
Max-Forwards — Используется для ограничения количества прокси или шлюзов, которые могут пересылать запрос. Значение по умолчанию - 70.

Пример SIP запроса INVITE

INVITE sip:xxxx@1.1.1.1:5060;user=phone SIP/2.0

From: "Calling User" <sip:yyyy@1.1.1.1:5060>;tag=m3l2hbp

To: <sip:xxxx@1.1.1.1:5060;user=phone>

Call-ID: 3ud04chatv9q13@1.1.1.1

…

Content-Length: 254

v=0

m=audio 50024 RTP/AVP 8 0 2 18

a=rtpmap:8 PCMA/8000

a=rtpmap:0 PCMU/8000

a=rtpmap:2 G726-32/8000/1

a=rtpmap:18 G729/8000

a=sendrecv

a=rtcp:50025

Пример SIP ответа на запрос INVITE

SIP/2.0 200 OK

Via: SIP/2.0/UDP 1.1.1.12:5060;rport=5060;

From: "Calling User" <sip:yyyy@1.1.1.1:5060>;tag=m3l2hbp

…

Content-Length: 170

v=0

o=root 1673140232 1673140232 IN IP4 1.1.1.1

m=audio 13504 RTP/AVP 8

a=rtpmap:8 PCMA/8000/1

a=candidate:4234997325 1 udp 2043278322 1.1.1.14 44323 typ host

a=ptime:20

a=sendrecv

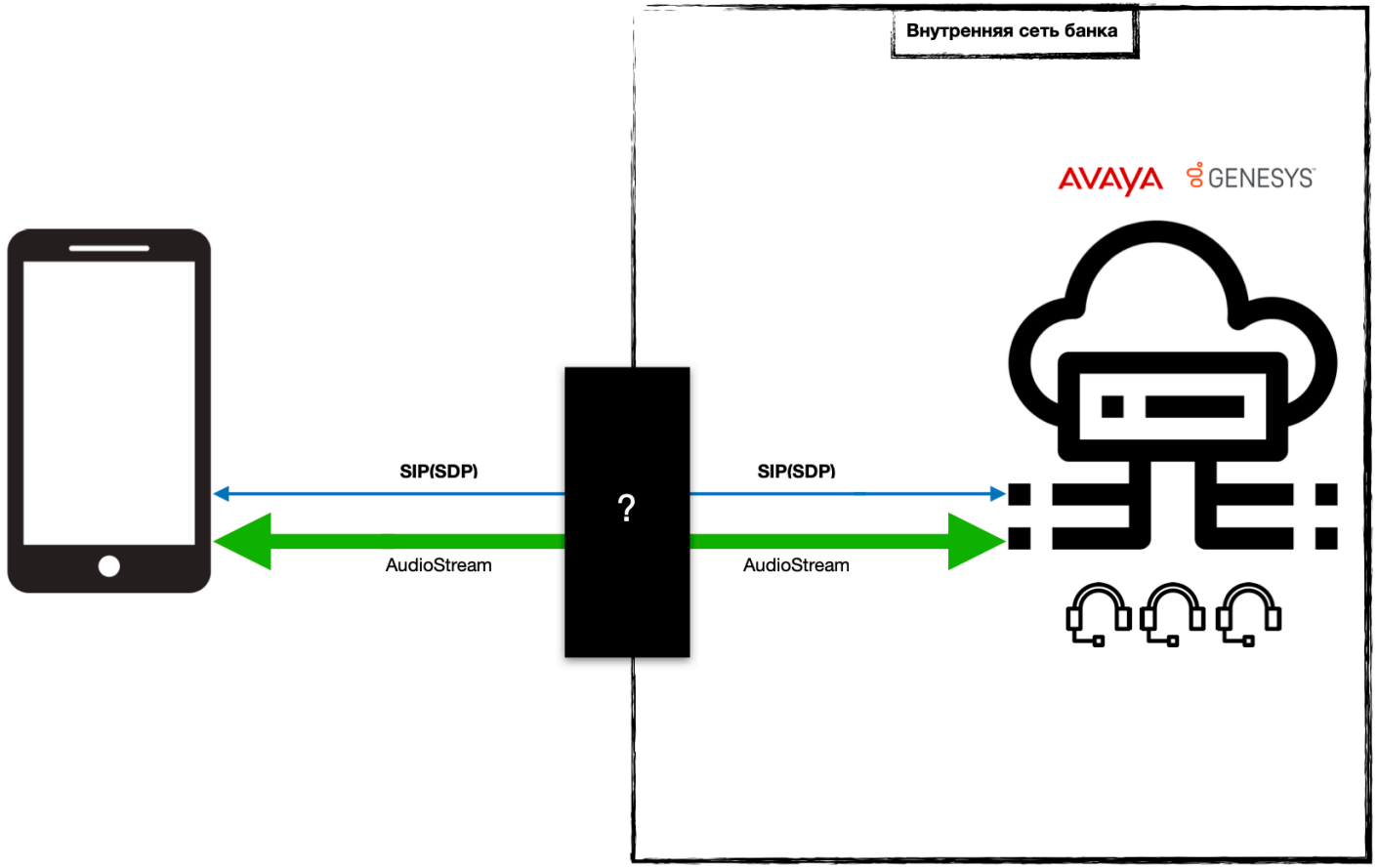
SDP протокол. Как часть SIP протокола.

SDP — это аббревиатура от Session Description Protocol — протокол описания сеанса связи. Session Description Protocol определяет параметры обмена мультимедийными данными (как правило, потоковыми) между двумя конечными точками.

Пример:

```
v=0
o=2398026505 2307593197 IN IP4 1.1.1.1
s=Audio Session
c=IN IP4 1.1.1.2
t=0 0
m=audio 15010 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=sendrecv
```

Отобразим изученные протоколы на нашей схеме



WebSocket

WebSocket (Веб-сокет) — это протокол полнодуплексной связи поверх TCP-соединения. То есть с помощью этого протокола можно передавать и принимать сообщение одновременно. Он позволяет в режиме реального времени обмениваться сообщениями между сервером и клиентом (браузером).

Используем WebSocket-соединение **для передачи текстовых запросов протокола SIP.**

Всегда предпочитайте **wss://**

Реализация WebSocket-соединения в iOS приложении

Решение от компании Apple:

URLSessionWebSocketTask(iOS 13.0+)

https://developer.apple.com/documentation/foundation/urlsessionwebsockettask

Сторонние библиотеки:

https://github.com/facebookincubator/SocketRocket

Реализация WebSocket-соединения в iOS приложении с помощью библиотеки SocketRocket

```
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
[request setTimeoutInterval:gConnectionTimeoutInterval];
request.networkServiceType = NSURLNetworkServiceTypeVoIP;

SRWebSocket *webSocket = [[SRWebSocket alloc] initWithURLRequest:request protocols:@[@"sip"]];
webSocket.delegate = self;
[webSocket open];
```

M

Реализация WebSocket-соединения в iOS приложении с помощью SocketRocket

```
#pragma mark SRWebSocketDelegate

- (void)webSocket:(SRWebSocket *)webSocket didReceiveMessage:(id)message {
    // обработка входящих SIP сообщений
}

- (void)webSocketDidOpen:(SRWebSocket *)webSocket {
    [self sendRegisterRequest];
}

- (void)webSocket:(SRWebSocket *)webSocket didCloseWithCode:(NSInteger)code reason:(NSString *)reason wasClean:(BOOL)wasClean {
    // обработка закрытия сокета
}

- (void)webSocket:(SRWebSocket *)webSocket didFailWithError:(NSError *)error {
    // обработка ошибок
}
```

Реализация WebSocket-соединения в iOS приложении с помощью SocketRocket

```
- (void)sendRegisterRequest {
    NSMutableString *req = [NSMutableString string];
    [req appendString:[NSString stringWithFormat:@"REGISTER %@ SIP/2.0\r\n", regUri]];
    [req appendString:[NSString stringWithFormat:@"Call-ID: %@\r\n",[self makeCallID]]];
    [req appendString:[NSString stringWithFormat:@"CSeq: %@\r\n", CSeq]];
    [req appendString:[NSString stringWithFormat:@"From: <sip:%@@%@>;tag=%@\r\n", gFromUserName, gKamailioAlias, [self makeFromTag]]];
    [req appendString:[NSString stringWithFormat:@"To: <sip:%@@%@>\r\n", gFromUserName, gKamailioAlias]];
    [req appendString:[NSString stringWithFormat:@"Via: SIP/2.0/UDP %@:%@;branch=%@\r\n", localIPAddress, gLocalPort, [self makeBranch]]];
    [req appendString:@"Accept: application/sdp\r\n"];
    [req appendString:@"Accept: text/plain\r\n"];
    [req appendString:@"Content-Length: 0\r\n"];
    [req appendString:@"\r\n"];

    [webSocket send:req];
}
```

SSL certificate pinning

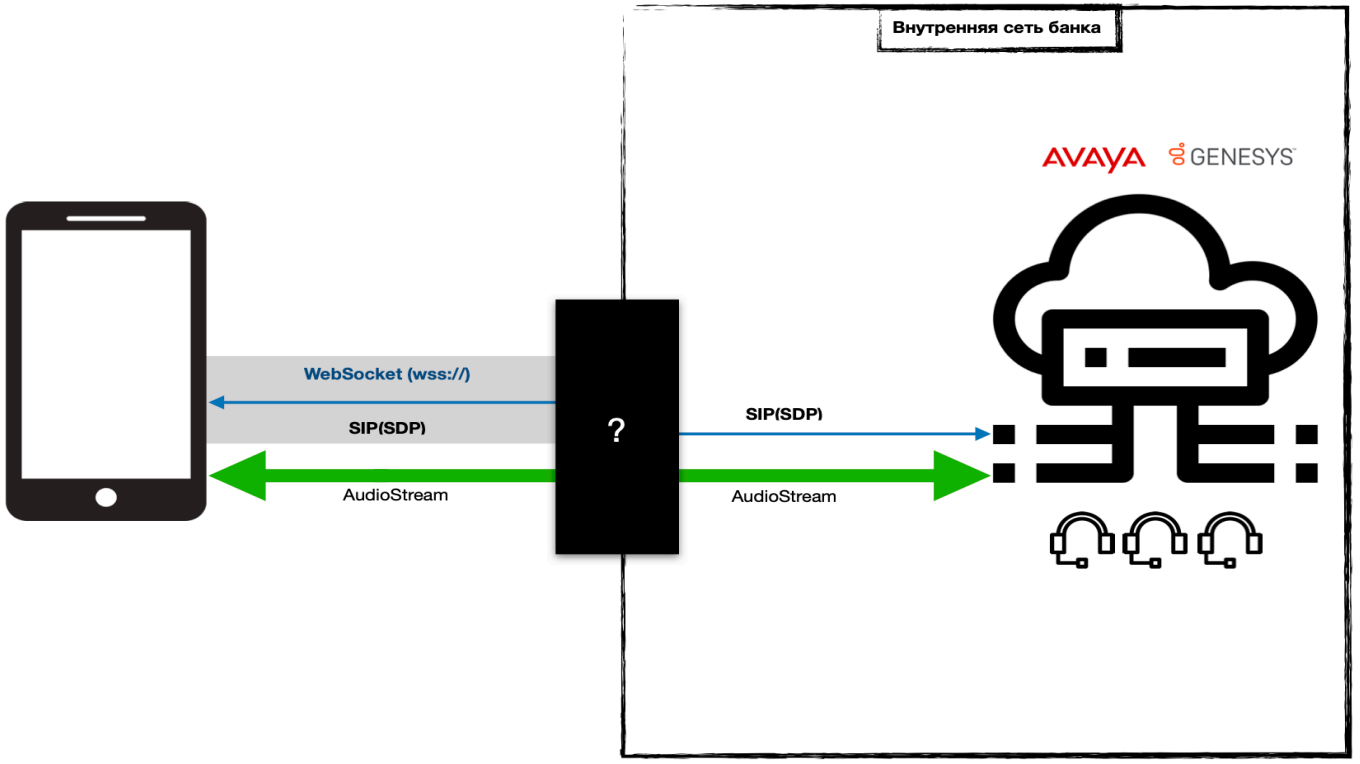
Совместно с протоколом wss используем **SSL certificate pinning**.

Certificate pinning – это внедрение SSL сертификата, который используется на сервере, в код мобильного приложения.

Реализация WebSocket-соединения в iOS приложении с помощью SocketRocket. Добавление SSL Pinning.

```
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
[request setTimeoutInterval:gConnectionTimeoutInterval];
request.networkServiceType = NSURLNetworkServiceTypeVoIP;

// --- SSL Pinning begin ---
id certificateRef = [self loadPinningCertificate:gSSLPinningCertName];

NSMutableArray *certificates = [NSMutableArray new];
[certificates addObject:certificateRef];
[request setSR_SSLPinnedCertificates:certificates];

// --- SSL Pinning end ---

SRWebSocket *webSocket = [[SRWebSocket alloc] initWithURLRequest:request protocols:@[@"sip"]];
webSocket.delegate = self;
[webSocket open];
```

Добавим WebSocket на схему



Протоколы передачи аудио и видео данных.

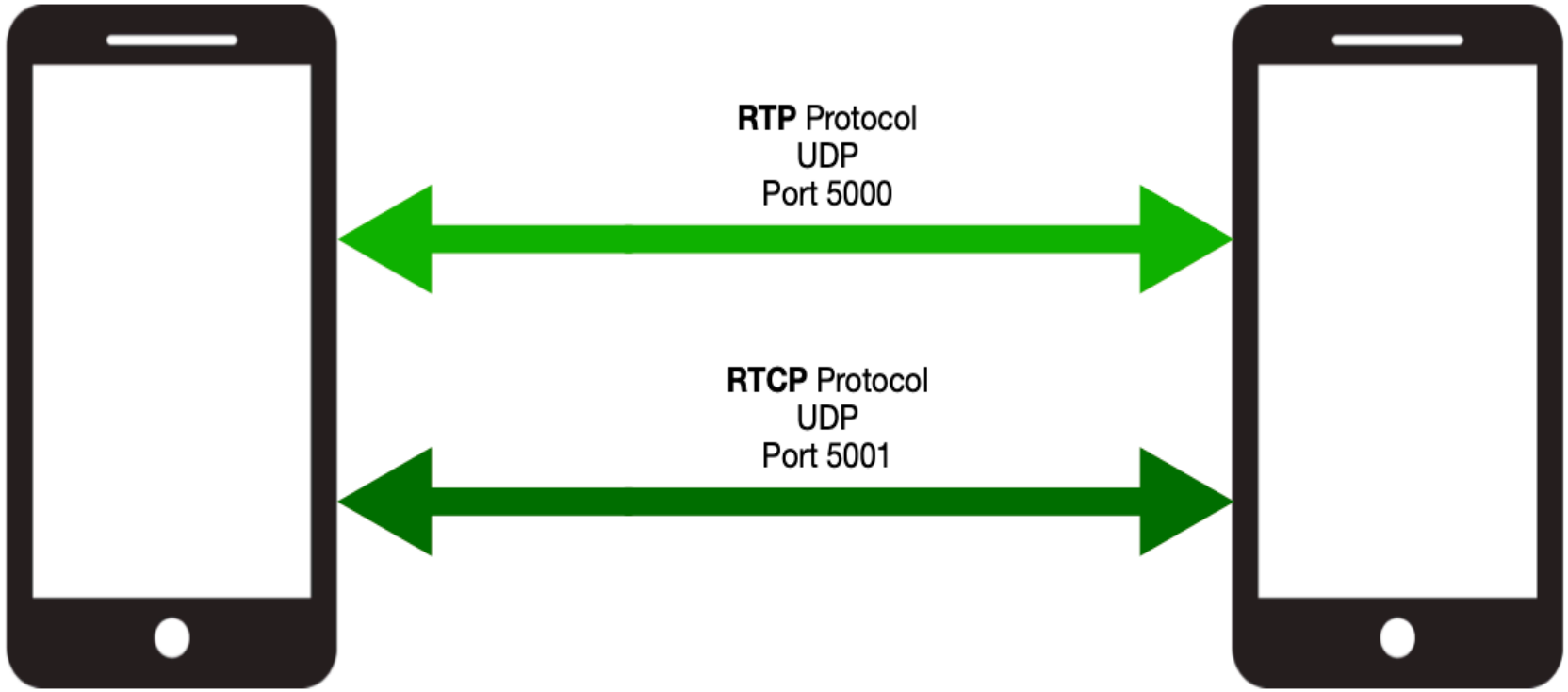
Протокол RTP (Real-time Transport Protocol) определяет стандарт пакетов для передачи мультимедиа-данных (аудио и видео) через Интернет.

RTP работает вместе с протоколом **RTP Control Protocol (RTCP)**.

Если RTP передает медиапоток, то RTCP используется для мониторинга параметров передачи потока и обеспечения и качества обслуживания (QoS), а также для синхронизации нескольких потоков.

Протокол SRTP (Secure RealTime Transport Protocol, безопасный протокол передачи данных в реальном времени) - это расширение протокола RTP, который добавляет дополнительные функции безопасности, такие как аутентификация сообщений, конфиденциальность и защита от прослушивания.

Схема работы RTP/RTCP протоколов



WebRTC

WebRTC (Web Real Time Communications) — это стандарт, который описывает передачу потоковых аудиоданных, видеоданных и контента между браузерами (без установки плагинов или иных расширений) или другими поддерживающими его приложениями в режиме реального времени.

Подробнее здесь https://webrtc.org/

Там же для сборок вручную приложены инструкции:

https://webrtc.googlesource.com/src/+/main/docs/native-code/index.md

Для iOS можно воспользоваться годичной давности спеками для cocoapods:

https://github.com/CocoaPods/Specs/tree/master/Specs/2/c/6/GoogleWebRTC

Схема работы WebRTC

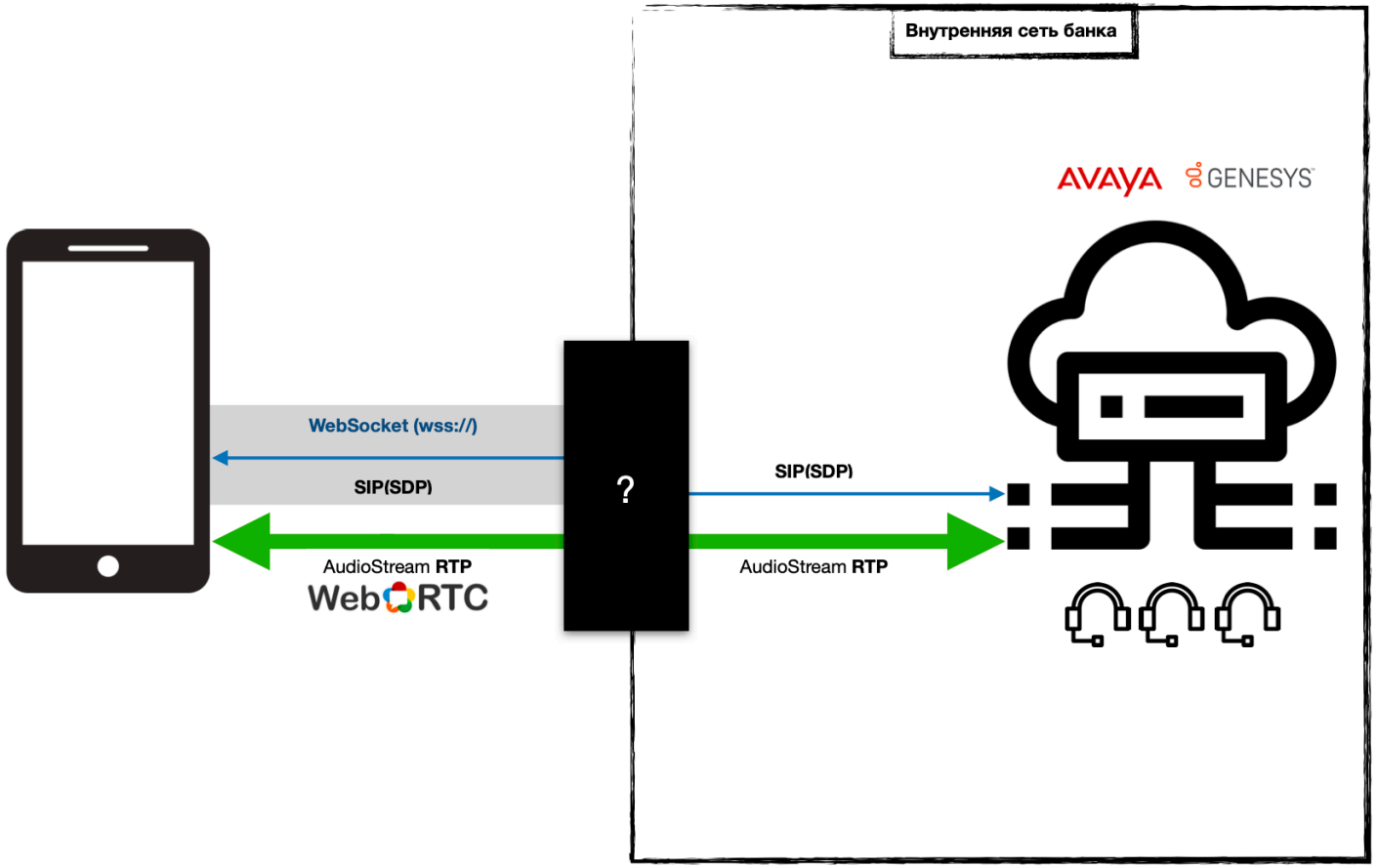
Подключение/Работа с WebRTC в iOS приложении

```
RTCPeerConnectionFactory *factory = [[RTCPeerConnectionFactory alloc] init];
RTCConfiguration *conf = [[RTCConfiguration alloc] init];
RTCPeerConnection *callPeerConnection = [factory peerConnectionWithConfiguration:conf constraints:[self offerConstraints] delegate:self];
RTCMediaStream *localStream = [factory mediaStreamWithStreamId:[self localStreamLabel]];
RTCAudioTrack *audioTrack = [factory audioTrackWithTrackId:[self audioTrackId]];
[localStream addAudioTrack:audioTrack];
[self.callPeerConnection addStream:localStream];

__weak typeof(self) weakSelf = self;
[_callPeerConnection offerForConstraints:[self offerConstraints] completionHandler:^(RTCSessionDescription * _Nullable sdp, NSError * _Nullable error) {
    dispatch_async(dispatch_get_main_queue(), ^{
        __strong typeof(weakSelf) strongSelf = weakSelf;
        [strongSelf.callPeerConnection setLocalDescription:sdp completionHandler:^(NSError * _Nullable error) {
            __strong typeof(weakSelf) strongSelf = weakSelf;
            strongSelf->mySdp = strongSelf.mySdp.sdp;
        }];
    });
}];
```

Подключение/Работа с WebRTC в iOS приложении

```
- (void)handleInviteResponse:(NSString *)message
{
    NSArray *mesgLines = [message componentsSeparatedByString:@"\n"];
    NSMutableArray *remoteSdpLines = [NSMutableArray array];
    for (NSString *line in mesgLines) {
        if ([line rangeOfString:@"a=candidate:"].location == 0) {
            NSRange range = [line rangeOfString:@"a="];
            if (range.location != NSNotFound) {
                NSUInteger pos = range.location + range.length;
                NSString *candidateSdp = [line substringWithRange:NSMakeRange(pos, line.length - pos - 1)];
                if (candidateSdp && candidateSdp.length > 0) {
                    RTCIceCandidate *candidate = [[RTCIceCandidate alloc] initWithSdp:candidateSdp sdpMLineIndex:0 sdpMid:@"audio"];
                    [media addIceCandidate:candidate];
                }
            }
        }
        [remoteSdpLines addObject:line];
    }

    RTCSessionDescription *description = [[RTCSessionDescription alloc] initWithType:RTCSdpTypeAnswer sdp:[remoteSdpLines componentsJoinedByString:@"\n"]];
    [self.callPeerConnection setRemoteDescription:description completionHandler:^(NSError * _Nullable error) {  }];
}
```

Добавим протокол RTP и WebRTC на схему



NAT’ы

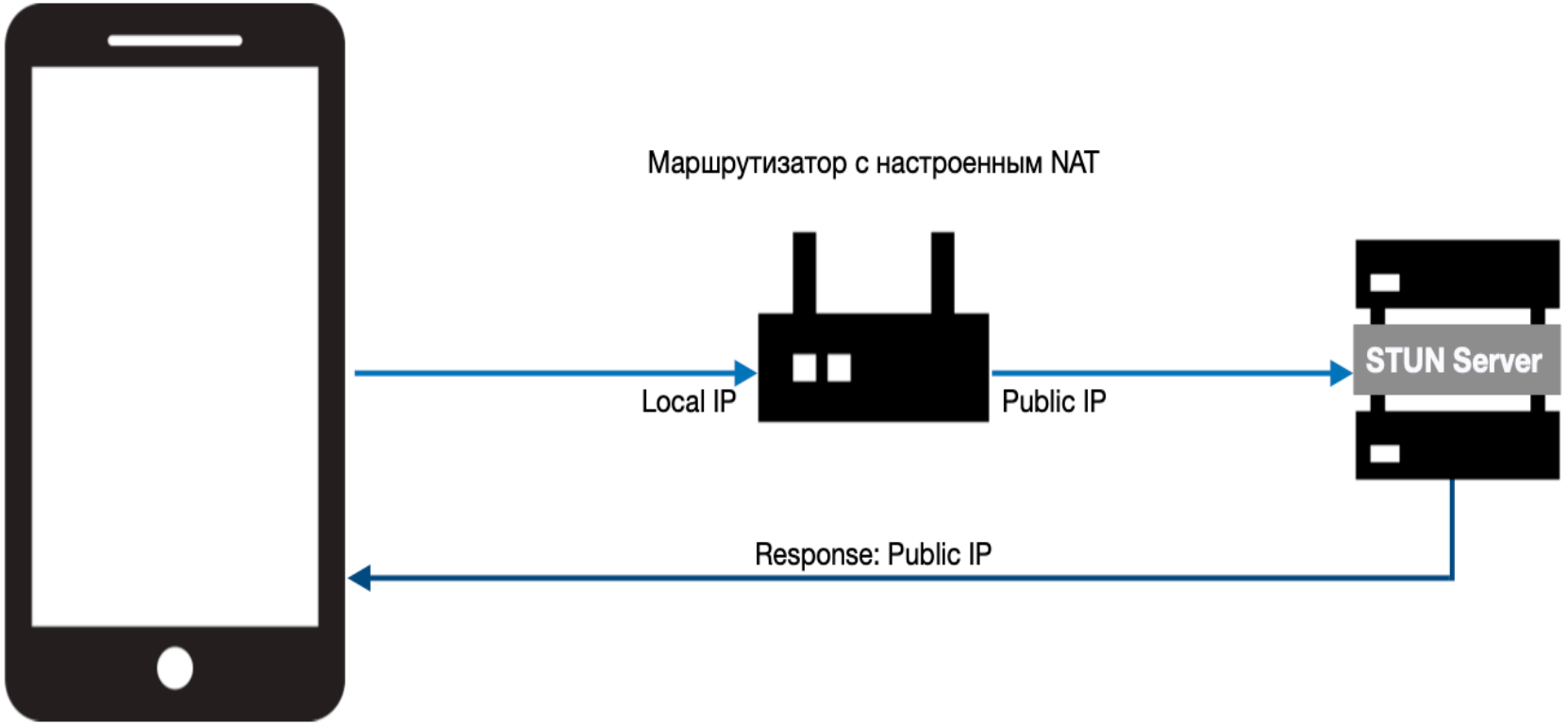
В Интернете исторически не хватало “номеров” для каждого подключенного устройства.

С IPv4 было доступно только около **4 миллиардов адресов**.

Недостаток адресов был решен путем группировки многих устройств под одним общим адресом с **маршрутизатором**, переводящим адреса в пакеты, проходящие через него.

Этот процесс называется **трансляцией сетевых адресов (NAT)**.

Протокол STUN — «Простое прохождение UDP через NAT»



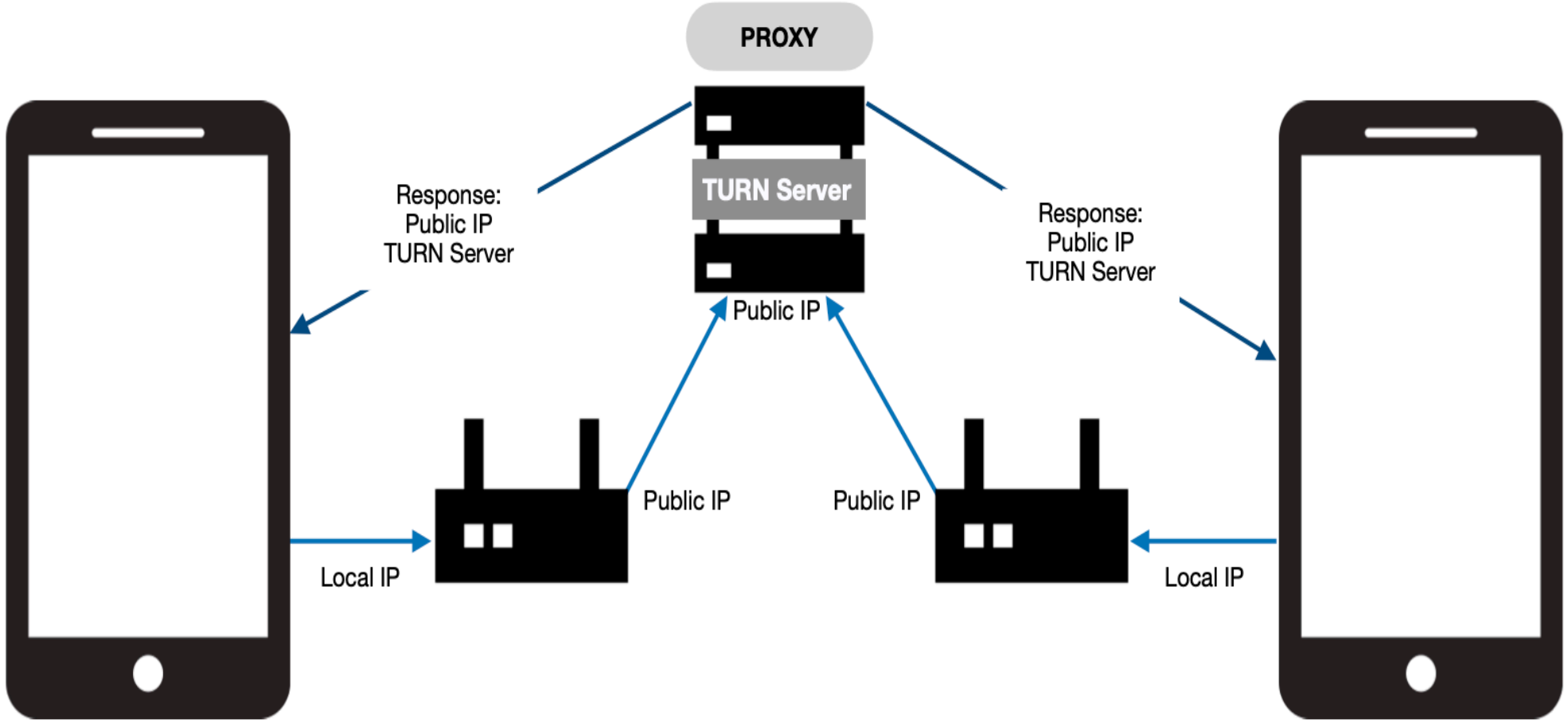
Протокол TURN

Предназначен для обхода ограничения "Симметричный NAT" путём открытия соединения с TURN сервером и ретрансляции всей информации через TURN сервер.

Принцип работы – разбиение симметричного NAT на два несимметричных.

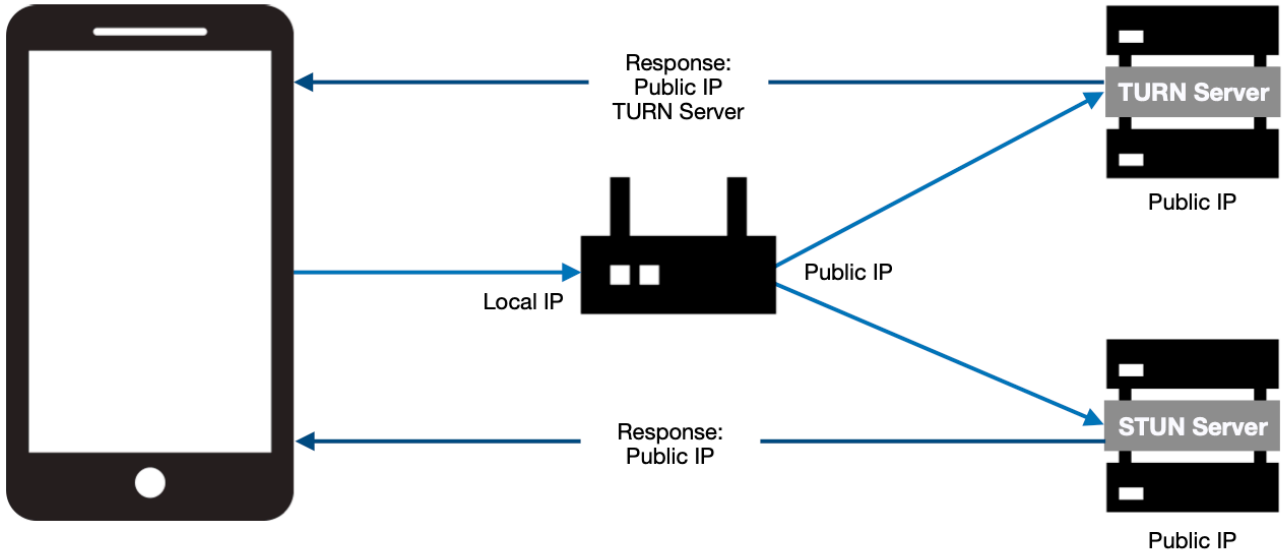
Клиент с TURN-агентом отправляет сообщение на TURN-сервер для того, чтобы определить public IP-адрес (по аналогии со STUN), но вместо этого TURN-сервер посылает свои IP и порт.

Схема работы протокола TURN



Протокол ICE

Протокол ICE - это дополнение (суб-протокол) к протоколам STUN и TURN.



Coturn, как пример реализации TURN сервера

Coturn - бесплатная реализация TURN и STUN Server с открытым исходным кодом.

Конфигурационный файл Coturn располагается - **/etc/coturn/turnserver.conf**

Подробнее здесь - https://github.com/coturn/coturn

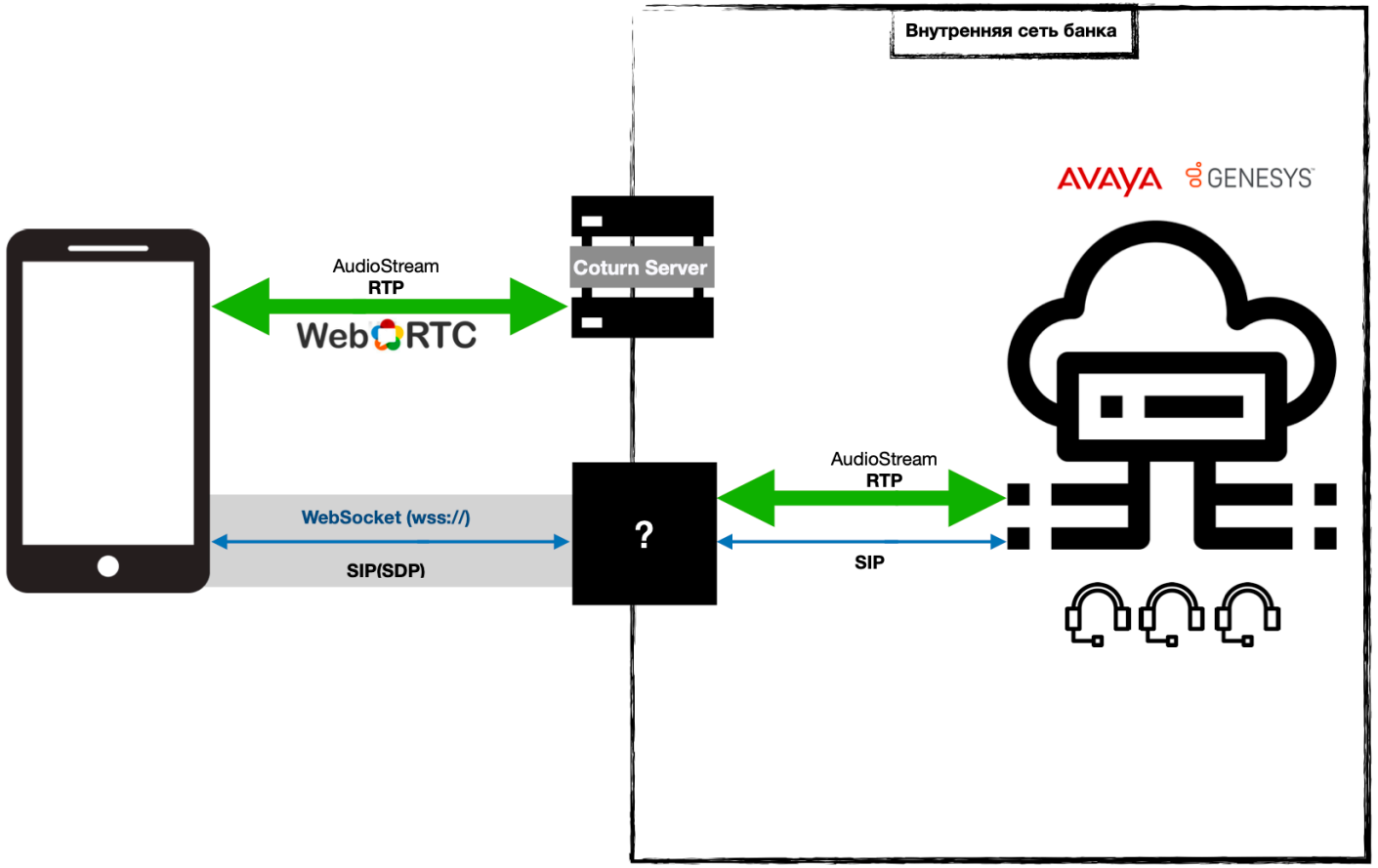
Листинг turnserver.conf

```
listening-port=xxxx  // Порт для прослушивания UDP и TCP (по умолчанию: 3478)
tls-listening-port=xxxx  // Альтернативный порт для прослушивания протоколов TLS и DTLS.
server-name=xxxx  // Имя сервера, используемое для цели аутентификации oAuth.
fingerprinting  // Использовать fingerprints в сообщениях TURN.
mobility  // Мобильность с поддержкой спецификаций ICE (MICE).
user=login:password  // Учетная запись пользователя
cert=/<path to file>/cert.pem  // Файл сертификата, формат PEM.
pkey=/<path to file>/key.key  // Файл закрытого ключа, формат PEM.
no-stdout-log  // Флаг для предотвращения сообщений журнала stdout.
min-port=50000  // Нижняя граница диапазона UDP-портов(Значение по умолчанию - 49152)
max-port=65000  // Верхняя граница диапазона UDP-портов(Значение по умолчанию - 65535)
```

Работа с STUN/TURN серверами в WebRTC в iOS приложении

```
RTCPeerConnectionFactory *factory = [[RTCPeerConnectionFactory alloc] init];
RTCConfiguration *conf = [[RTCConfiguration alloc] init];

// --- Add Stun/Turn servers begin
conf.iceServers = @[ [[RTCIceServer alloc] initWithURLStrings:@[[NSString stringWithFormat:@"stuns:%@:%@",gStunIpAddress, gStunPort]]],
                     [[RTCIceServer alloc] initWithURLStrings:@[[NSString stringWithFormat:@"turn:%@:%@",gTurnIpAddress, gTurnPort]]] ];
// --- Add Stun/Turn servers end

RTCPeerConnection *callPeerConnection = [factory peerConnectionWithConfiguration:conf constraints:[self offerConstraints] delegate:self];
```

Добавим Coturn на схему



Прокси Сервер

Прокси Сервер — компонент, выполняющий посреднические функции, который выступает в роли как сервера, так и клиента, в целях создания запросов от лица других клиентов.

SIP прокси сервер — это участник на пути маршрутизации SIP запросов к серверу пользователя (user agent servers) и доставки SIP ответов обратно к пользовательскому агенту (user agent clients).

Запрос может пройти через несколько прокси серверов на своем пути, до того как он достигнет АТС.

SIP-сервер Kamailio

Использование SIP-сервера Kamailio как проксирование звонка во внутреннюю сеть

Одной из выдающихся особенностей Kamailio является способность обслуживать большое количество активных пользователей в одном сервере (в зависимости от аппаратного обеспечения их может быть до 100 000+).

Длительный срок развития Kamailio обеспечивает стабильность, необходимую в режиме реального времени телекоммуникаций и широкий набором функций в работе с SIP сигнализацией.

Надежный, гибкий и отказоустойчивый SIP-сервер.

Подробнее здесь - https://www.kamailio.org/w/

Функции Kamailio

Kamailio выполняет:

- аутентификацию пользователей
- регистрацию пользователей
- сохранение местоположения пользователей
- маршрутизацию вызовов
- балансировку нагрузки между медиа серверами
- поддержку WebSocket для WebRTC, IPv4 и IPv6.
- сокрытие топологии и проксирование медиа потоков

Kamailio считывает данные конфигурации из **/etc/kamailio/kamailio.cfg**

kamailio.cfg

```
#!KAMAILIO
# Kamailio (OpenSER) SIP Server v4.0 - default configuration script

####### Defined Values #########

#!define FLT_ACC 1
#!define FLT_ACCMISSED 2

....

####### Global Parameters #########

debug=2
log_stderror=no
enable_tls=yes

....

####### Modules Section ########

loadmodule "textops.so"
loadmodule "siputils.so"
loadmodule "acc.so"

....

####### Setting module-specific parameters #########

# ----- acc params -----
modparam("acc", "early_media", 0)
modparam("acc", "report_ack", 0)

....

####### Routing Logic ########


# Main SIP request routing logic
request_route {

....

}

# Other logic

....

#!endif
```

Основные модули Kamailio

dispatcher

Websocket

Tls

Rtpengine

Uac

Htable

http_client

Модули Kamailio - https://kamailio.org/docs/modules/

Модуль dispatcher

Этот модуль предлагает функции балансировки нагрузки SIP и может использоваться в качестве диспетчера трафика SIP.

Он очень легкий, поэтому подходит для обработки тяжелого SIP-трафика. Модуль занимает мало места и может загружать правила балансировки из обычного текстового файла.

Этот модуль мы использовали для проксирования трафика на внутреннюю АТС банка.

Конфигурация диспетчеризации(проксирования) берется из файла **/etc/kamailio/dispatcher.list**

Его формат достаточно прост:

1 sip:1.1.1.1:5060

2 sip:1.1.1.2:5060

Модуль websocket

Этот модуль реализует сервер WebSocket (RFC 6455) и обеспечивает установление соединения (подтверждение связи), управление (включая поддержание активности соединения)

Модуль tls

Этот модуль реализует транспорт TLS для Kamailio с помощью библиотеки OpenSSL (http://www.openssl.org).

Настройка модуля(**/etc/kamailio/kamailio.cfg**):

```
loadmodule "modules/tls/tls.so"

modparam("tls", "private_key", "./key.pem")
modparam("tls", "certificate", "./cert.pem")
modparam("tls", "ca_list", "./calist.pem")

enable_tls=yes
```

Модуль rtpengine

Это модуль, который позволяет передавать медиапотоки через прокси-сервер RTP .

Это Sipwise rtpengine https://github.com/sipwise/rtpengine.

Модуль rtpengine - это модифицированная версия исходного модуля rtpproxy, использующая новый протокол управления.

Модуль работает только с прокси-серверами RTP, которые специально его поддерживают.

Модуль UAC

UAC (User Agent Client) предоставляет некоторые базовые функции UAC, такие как отправка SIP-запросов, регистрация в удаленной службе, обработка заголовка From: (анонимизация) и аутентификация клиента.

Мы активно использовали методы замены в SIP заголовке From/To части URI:

```
uac_replace_from("sip:$var(new_from)@" + "$rd");
uac_replace_to("sip:$var(new_to)@" + "$rd");
```

Модуль htable

Модуль добавляет возможность использования хеш-таблицы.

Очень помогает если нужно сохранять некие параметры для всего времени жизни разговора.

Использование (**/etc/kamailio/kamailio.cfg**):

```
modparam("htable", "htable", "buffer=>size=8;")
...
$sht(buffer=>test) = 1;
```

Модуль http_client

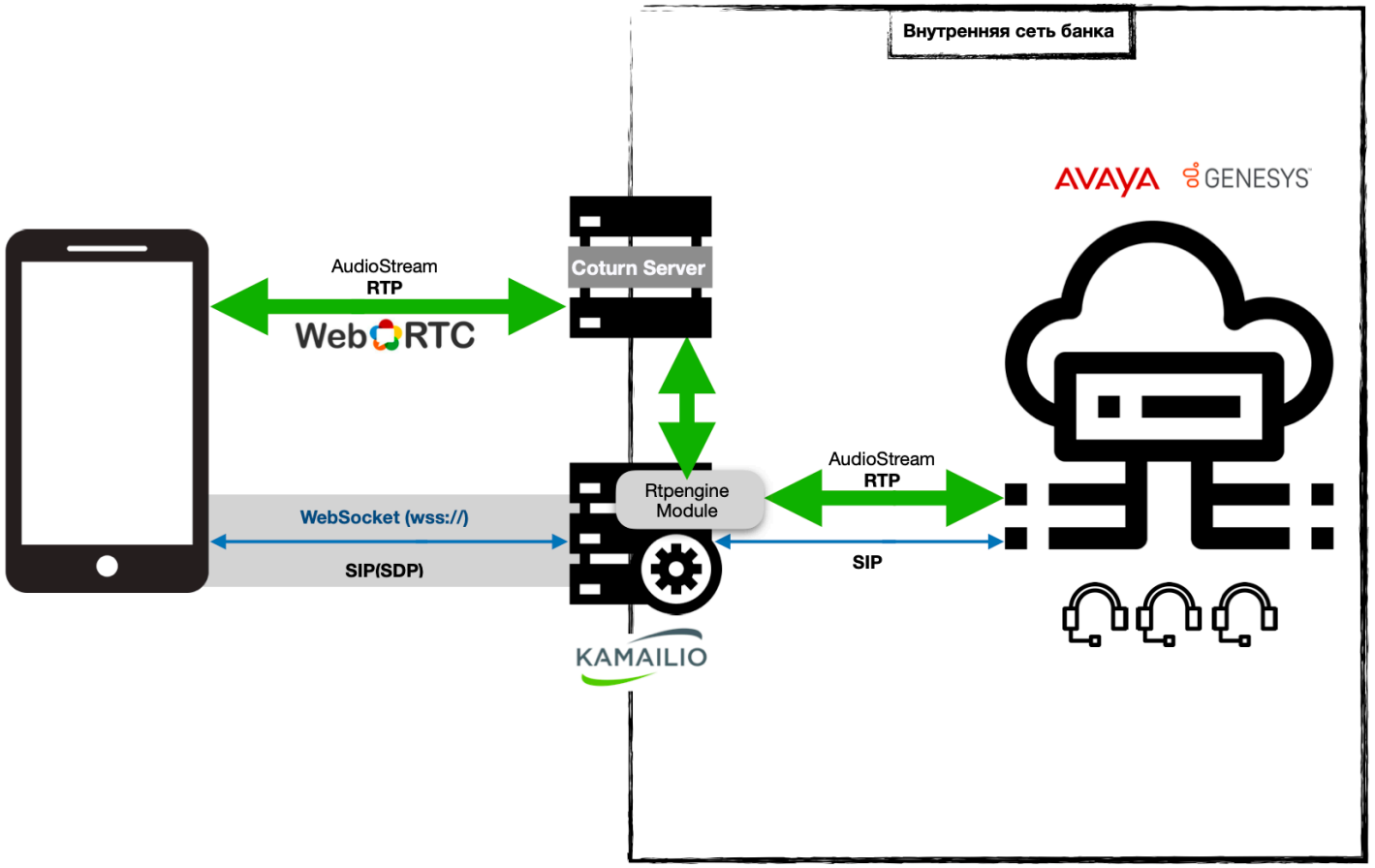
Этот модуль реализует функции протокола, которые используют библиотеку **libcurl** для получения данных с внешних HTTP-серверов или отправки данных на HTTP-серверы.

Функция http_client_query позволяет Kamailio выдавать HTTP-запрос GET и получать доступ к частям ответа.

Пример (**/etc/kamailio/kamailio.cfg**):

```
$var(http_code) = http_client_query("https://site.com", "$var(result)");
if ($var(http_code) == 200) {
    $sht(buffer=>data) = $avp(result);
}
```

Добавим Kamailio к схеме - Финальный Результат!



МКБ

Спасибо за внимание!