30-04-2016

SA8T2 Internal Deliverable STUN and TURN: how WebRTC deals with Firewalls and NAT

SA8T2 Internal Deliverable

Contractual Date:	30-04-2016
Actual Date:	30-04-2016
Grant Agreement No.:	691567
Activity:	12/SA8
Task Item:	Task 2 – WebRTC
Nature of Deliverable:	R (Report
Dissemination Level:	PU (Public)
Lead Partner:	NORDUnet (UNINETT)
Authors:	Mihály Mészáros (NIIFI)

The research leading to these results has received funding from the European Union’s Horizon 2020 research and innovation programme under Grant Agreement No. 691567 (GN4-1).

Abstract
This document reports on results and findings from a technical investigation into the use of the IETF ICE, STUN and TURN protocols for NAT and firewall traversal with WebRTC, in the context of European higher education and research. The technology scout was conducted by the Service Activity 8 (SA8, Real Time Communication and Media), Task 2 (WebRTC) team as part of the GN4-1 project. This report should, as such, be read in context of the related work produced by GN4-1 SA8-T2.

Table of Contents

1 Introduction 3

1.1 About this document 3

1.1.1 Target audience 3

1.1.2 Responsible task members 3

1.2 Background 3

1.3 Rationale 4

1.4 Tech scout objective and methodology 4

2 Technology and deployments 5

2.1 ICE, STUN and TURN protocol history 5

2.2 ICE and WebRTC 6

2.3 Market exploration 6

3 PoC distributed STUN/TURN service 8

3.1 PoC overall setup goals 8

3.2 Credential mechanisms 8

3.2.1 Long Term Credential Mechanism (username, password, realm based) 8

3.2.2 Time Limited Long Term Credential Mechanism (REST API) 8

3.2.3 OAuth 9

3.3 Ansible for automated node deployment 10

3.4 PoC Overview 10

3.5 Setup of the PoC hosts 11

3.5.1 VM images 12

3.5.2 Installation and configuration of the PoC Hosts 12

3.6 Database Schemas and extensions 14

3.6.1 Database schema for Long Term Credential 14

3.6.2 Database schema for REST support 15

3.7 PoC Web Interface and Authentication schemas 17

3.8 REST API 17

3.9 Geolocation for load-sharing and load-balancing 17

3.9.1 Geolocation for Long Term Credential (LTC) 17

3.9.2 Geolocation for the Time Limited Long Term Credential (REST) 18

4 Lessons learned 19

4.1 Technology strength 19

4.2 Technology shortcomings 19

4.3 STUN/TURN service meets R&E community needs 20

5 Conclusion 22

Table of Figures

Figure 1: STUN/TURN service overview 11

Figure 2: Table `turnusers_lt` 15

Figure 3: Database structure and relations 16

1 Introduction

1.1 About this document

This report covers the results of a technical investigation into the use of the IETF ICE, STUN and TURN protocols for NAT and firewall traversal with WebRTC, in the context of European higher education and research. The first part of the document provides a technical overview of ICE, STUN and TURN in WebRTC. The second part presents a proof of concept implementation of a distributed ICE, STUN and TURN infrastructure for European higher education and research.

The technology scout was undertaken by the Service Activity 8 (SA8, Real Time Communication and Media), Task 2 (WebRTC) team as part of the GN4-1 project. This report should, as such, be read in context of the related work produced by GN4-1 SA8-T2.

The WebRTC task ran from 1 May 2015 to 30 April 2016.

1.1.1 Target audience

This document targets technical management and specialists, in particular those working in the fields of real time communications, eLearning and eResearch.

1.1.2 Responsible task members

Mihály Mészáros (NIIF) had the lead on this tech scout. Jan Meijer (UNINETT) and Simon Skrødal (UNINETT) were the document editors.

1.2 Background

ICE (Interactive Connectivity Establishment) is a protocol for Network Address Translator (NAT) traversal for UDP-based multimedia sessions. It allows audio and video streams to flow between communication end points despite hindrances like NATs and firewalls. ICE makes use of the Session Traversal Utilities for NAT (STUN) protocol and its extension, Traversal Using Relay NAT (TURN). ICE can be used by any protocol utilizing the offer/answer model such as WebRTC.

The ICE, STUN and TURN protocols are defined by the IETF. They are mandatory to implement for WebRTC-compliant endpoints. From statistics gathered by WebRTC statistics cloud service provider callstats.io we know that without these protocols more than 10% of WebRTC talks would fail because an audio and/or video stream could not be set up. For the end user this typically manifests itself by being unable to hear/see other participants, or be seen/heard by the other participants, which leads to "It doesn't work" frustration. ICE, STUN and TURN prevent this unfortunate user experience from happening.

1.3 Rationale

Most R&E users encounter firewalls and NATs on campus, at home and on the road. These middle-boxes cause problems for real time communication data streams (audio, video, data). The ICE, STUN and TURN protocols are the widely accepted open standards to address and solve complex problems with NAT/firewall traversal and IPv6 transitioning and are mandatory to implement in WebRTC-compliant end points.

Any WebRTC deployment in R&E will therefore need to address how it deals with ICE, STUN and TURN. This technology scout provides the technology background for a recommendation on an ICE, STUN and TURN infrastructure recommendation for the European R&E community.

1.4 Tech scout objective and methodology

The technology scout started with desk research into STUN/TURN technology standards, available products and services and how this relates to WebRTC. After this phase it was clear that anyone who would want to deploy WebRTC services or applications (especially peer to peer based) or who'd be deploying gateway boxes would quickly have a need for a STUN/TURN service. Such a service would have to be procured or built.

To facilitate both processes a proof of concept STUN/TURN service supporting both WebRTC applications and legacy SIP devices was built. The goal with the PoC was threefold:

- explore the use of federated authentication to allow WebRTC applications to authenticate to a STUN/TURN service
- explore the feasibility of building a distributed STUN/TURN service in the GÉANT community
- gain practical experience with STUN/TURN server technology for use in the requirements specification process of a possible procurement

For the proof of concept, the most popular and mature open source software product was used: CoTURN. Amongst others Google uses CoTURN.

2 Technology and deployments

2.1 ICE, STUN and TURN protocol history

ICE, STUN and TURN are IETF protocols used to ensure multimedia applications can deal with "middle boxes" like NATs and firewalls. ICE (Interactive Connectivity Establishment) is the protocol for Network Address Translator (NAT) traversal for UDP-based multimedia sessions. ICE makes use of the STUN (Session Traversal Utilities for NAT) protocol and its extension, TURN (Traversal Using Relay NAT). ICE can be used by any protocol utilizing the offer/answer model, like SIP or WebRTC.

The history of ICE, STUN and TURN started with STUN. STUN (in its first incarnation short for Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)) was published as RFC 3489 by the IETF in 2003 as an answer to the problems associated with UDP NAT and firewall traversal. Meant as a complete solution the protocol specification also states

> *This protocol is not a cure-all for the problems associated with NAT. It does not enable incoming TCP connections through NAT. It allows incoming UDP packets through NAT, but only through a subset of existing NAT types.*

Field experience with RFC 3489 showed this observation to be correct and the subset not to be sufficient to adequately address several real life scenarios. A better solution was needed. The IETF MMUSIC Working Group therefore developed a new, more complex standard that aimed to address the shortcomings of the "classic" STUN as specified in RFC 3489. The standardization process lasted 5 years and in 2010 these three standards were produced:

- ICE (Interactive Connectivity Establishment), RFC 5245, a complex protocol on the client side
- a new STUN (Session Traversal Utilities for NAT), RFC 5389
- TURN (Traversal Using Relay NAT), RFC 5766

Although the combination of client-side ICE and server-side STUN/TURN services aims to address all of our complex issues with traversal of NAT, firewalls and other middle boxes, a number of smaller extensions have appeared.

STUN/TURN are umbrella protocols and have many extensions (e.g. DTLS, TCP, IPv6, Origin, REST, OAuth, Bandwidth, etc.) beyond their core (RFC 5389, RFC 5766). It is a living and evolving standard and, as mentioned, the only widely accepted solution deployed to deal with the complex NAT/firewall traversal and next generation IPv6 smooth transition problem space.

2.2 ICE and WebRTC

The current Internet-Draft for WebRTC transports specifies ICE, STUN and TURN MUST be supported by compliant WebRTC end points. The specification reads as follows:

```
3.4.  Middle box related functions

The primary mechanism to deal with middle boxes is ICE, which
is an appropriate way to deal with NAT boxes and firewalls
that accept traffic from the inside, but only from the outside
if it is in response to inside traffic (simple stateful
firewalls).

ICE [RFC5245] MUST be supported.  The implementation MUST be a
full ICE implementation, not ICE-Lite.  A full ICE
implementation allows interworking with both ICE and ICE-Lite
implementations when they are deployed appropriately.

In order to deal with situations where both parties are behind
NATs of the type that perform endpoint-dependent mapping (as
defined in [RFC5128] section 2.4), TURN [RFC5766] MUST be
supported.

WebRTC browsers MUST support configuration of STUN and TURN
servers, both from browser configuration and from an
application.

In order to deal with firewalls that block all UDP traffic,
the mode of TURN that uses TCP between the client and the
server MUST be supported, and the mode of TURN that uses TLS
over TCP between the client and the server MUST be supported.
See [RFC5766] section 2.1 for details.

In order to deal with situations where one party is on an IPv4
network and the other party is on an IPv6 network, TURN
extensions for IPv6 [RFC6156] MUST be supported.

                    https://tools.ietf.org/html/draft-ietf-rtcweb-transports-12
                                                 expires September 22, 2016
```

ICE technology is unique in its manner. In the negotiation between WebRTC end points it will by default try to create the shortest and most direct way between the communication peers. Less hops between the peers means minimal delay and latency, providing a substantially improved real-time experience for the end users.

2.3 Market exploration

We explored the STUN/TURN product and services market and technology possibilities in order to understand the WebRTC ecosystem and landscape. This yielded multiple STUN/TURN server software implementations (listed below), though we were surprised to find less commercial services than expected. We found a number of open source standard implementations, with various quality and stability.

Open source server implementations:

- http://sourceforge.net/projects/stun/
- http://turnserver.sourceforge.net/
- https://github.com/jitsi/turnserver
- https://www.resiprocate.org/ReTurn_Overview
- http://www.creytiv.com/restund.html
- https://github.com/coTURN/rfc5766-turn-server/
- https://github.com/coTURN/coTURN

Commercial server implementations:

- http://www.eyeball.com/products/stun-turn-server/
- http://help.estos.com/help/en-US/procall/5/erestunservice/dokumentation/index.htm

Commercial Services:

- http://xirsys.com/
- https://www.twilio.com/stun-turn

The commercial services typically charge for bandwidth relayed through TURN connections.

3 PoC distributed STUN/TURN service

3.1 PoC overall setup goals

We wanted to build a Proof of Concept of a distributed STUN/TURN service with nodes close to the end-user leveraging the GÉANT network footprint. The PoC needed to provide STUN/TURN server backend functionality for WebRTC-enabled web applications and appliances supporting both WebRTC and other RTC protocols, notably SIP.

To facilitate the objectives of this technology scout, we used open components only. This provided the best opportunity to study all components comprising the service, test and implement various integrations and avoided vendor lock-ins. It also provides an easy path from PoC to pilot service for the GÉANT community should such a decision be made.

Last but not least WebRTC is all about the flexibility given by the marriage of open standards and open components. Accordingly, we endeavoured to build it as transparent as possible, making all the building blocks of the service public and open. All third party components used were open source as well.

3.2 Credential mechanisms

Our pilot supports the two most widely used authentication mechanisms. We also investigated a third option, OAuth, but unfortunately it is not yet implemented by browser vendors.

3.2.1 Long Term Credential Mechanism (username, password, realm based)

The original credential authentication mechanism for STUN/TURN, and still the most widely deployed today. We used it to support legacy VoIP terminals, like hardphones and softphones, (e.g. csipsimple on Android) and video-conference endpoints (e.g. cisco "C series"). Many other services and appliances only support this mechanism. It is defined in RFC 5389, section 10.2 (https://tools.ietf.org/html/rfc5389 - section-10.2)

3.2.2 Time Limited Long Term Credential Mechanism (REST API)

This mechanism addresses problems and issues detected in the original method, but was above all designed for Web/WebRTC purposes (where hiding a long term password credential is difficult or impossible). It is defined in https://tools.ietf.org/html/draft-uberti-rtcweb-turn-rest-00. This internet-draft is no longer being worked on, the concepts have been absorbed by the work on the OAUTH credential mechanism (see section 3.2.3 OAuth). The mechanism specified in draft-uberti-rtcweb-turn-rest-00 is nonetheless currently implemented in browsers whereas the new OAuth mechanism isn't yet.

The Time Limited Long Term Credential Mechanism tries to avoid the problem with username tracking by using a non-mandatory application specific string concatenated with a colon sign and with a timestamp. This authentication mechanism provides a backend REST API service for WebRTC applications and services. API Access control is provided by an API key/token that could be requested and obtained from a self-service portal. The API and the API key/token provide on the fly access to time limited credentials, which may in turn be set in the WebRTC/ICE engine of the end user’s web browser. The WebRTC application can thus forward information about the client (location) IP address to the REST API. By using a GeoIP database, we can ascertain the client’s location and offer the closest STUN/TURN service. As a result, this will likely achieve the lowest possible latency between peers.

Secrets are shared between the REST API (that provides time limited long term credentials) and the STUN/TURN servers (where these credentials are validated). According to https://tools.ietf.org/html/draft-uberti-rtcweb-turn-rest-00, the credentials provided by the API should only be valid for a limited time. In our case it is set to one day and for extra safety, the keys are rotated. This mechanism provides another time limit, so all shared keys older than two days are deleted from STUN/TURN servers and API servers. Thus, STUN/TURN servers cannot validate credentials older than 2 days since the shared key is no longer available. The key rotation mechanism adds a hard time to live limit, and prevents attacks based on manipulation of the STUN/TURN server’s clock or ntp settings. The clock and ntp setup is therefore very important on both sides (REST API and STUN/TURN server).

3.2.3 OAuth

We have not tested the OAuth mechanism, since we are not aware of any client browser implementation that supports it. We did, however, find one feature request issue about it on chromium.org (https://goo.gl/Z69q6I), and after we approached Mozilla enquiring about implementation status, they also opened an issue about this feature (https://goo.gl/6n78rL).

We also investigated possibilities of server side implementations. The chosen STUN/TURN implementation, coTURN, has some support for OAuth token validation on the server side. It is not sufficient, however, since we also need a tool to issue self-contained OAuth tokens.

In our search, we could not find any PHP library that supports the Authenticated-Encryption with Associated-Data (AEAD) scheme. We did, however, find some code samples on the OpenSSL website (https://goo.gl/UfuqTr).

The coTURN server source code contains a coTURN client library (written in C). It is based on OpenSSL and even more it is AEAD part is based on the abovementioned OpenSSL samples. The client library in file src/client/ns_turn_msg.c contains a nice function that could be reused in later phase of this pilot to set up a utility that could issue OAuth self-contained access tokens with AEAD;

```
int encode_oauth_token(const u08bits *server_name, encoded_oauth_token
*etoken, const oauth_key *key, const oauth_token *dtoken, const u08bits
*nonce)
```

Our conclusion is therefore that, when the browser implementations are made available, we could easily extend our STUN/TURN service PoC to support OAuth as well.

3.3 Ansible for automated node deployment

Ansible is a very handy tool that allows us to write playbooks to set up and manage identical operating system installations and service configurations, and to replicate these with ease. Playbooks provide clean and self-documenting installations and steps.

Debian's stable Ansible package, v.1.6-1.7, was missing desirable functionalities. We therefore opted to use the latest version (v.1.9+).

All playbooks are available on the following public GitHub repository: https://github.com/misi/stun-ansible

Please note that, while our playbooks are functional, they are not fully polished and finished. They were developed to demonstrate and validate the concept of using Ansible for automated node deployment. Quality improvement of these playbooks could be a work item in a next phase, after this Proof of Concept.

3.4 PoC Overview

We installed two groups of servers based on the authentication credential type (LTC and REST). One central node, brain.lab.vvc.niif.hu, provides multiple services and contains the central master MySQL database for the supported authentication methods.

Between the "brain", the master MySQL database and the coTURN servers (MySQL slaves) sits a secure encrypted MySQL master-slave replication. The central server runs a web server service with user login provided by the eduGAIN AAI. This site is the entrance to the STUN/TURN pilot service; it gives general information about the service and provides a self-service portal where the end user may request access (username/password or api_key) to the service. The central host also provides the REST API service, the REST API documentation and test site based on Swagger UI.

The overview image of the STUN/TURN service, below, depicts the Long Term Credential (LTC) and the REST (Time Limited Long Term Credential) services.

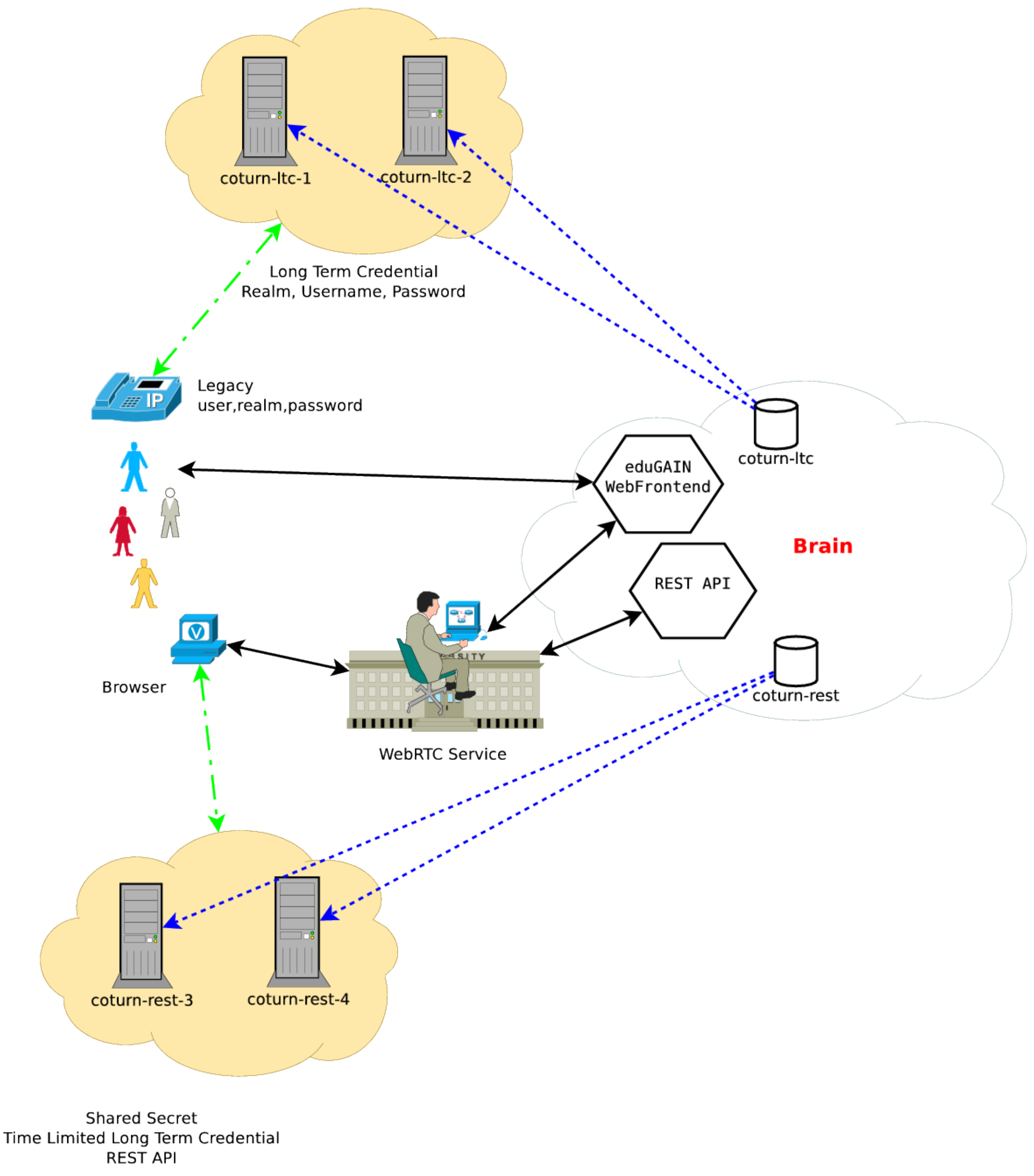

Figure 1: STUN/TURN service overview

3.5 Setup of the PoC hosts

For the coTURN instances and the central node, virtual machines with Debian Stable (Jessie) were used as the basis on which several packages were installed and configured.

3.5.1 VM images

For the coTURN instances, we created minimal Virtual Machines (VM), specifically:

- Memory: 500M physical, 500M swap
- Disk: 5GB disk
- CPU: 1 vCPU
- OS: Debian Stable (Jessie)
- Architecture: 64 bit

For the central node with web servers and more functionalities, we doubled the memory, but otherwise used almost the same setup:

- Memory: 1G physical, 500M swap
- Disk: 5GB disk
- CPU: 1 vCPU
- OS: Debian Stable (Jessie)
- Architecture: 64 bit

3.5.2 Installation and configuration of the PoC Hosts

During the setup, we configured and installed the packages as outlined below. For more details, please read the Ansible playbooks.

Configuration step-by step:

- Install basic packages
- Install SSH service
- Add admin user
    - Install SSH authorized keys
- Setup sudo
    - without password (use for user auth SSH-key pair)
    - keep auth socket
- Setup hosts and hostname, and domain name
- Setup mailname, exim4 mail server
- Install NTP time protocol
- Setup timezone
- install and setup ferm, so setup basic netfilter (the linux packet filter)
- Setup network, static IPv4, and static global IPv6 address, default gateway, network mask
- Setup DNS, two IPv4 and two IPv6 DNS resolver.
- Install TLS certificate
    - Create a PKI group, and setup ownership accordingly
    - Converting TLS private key to PKCS1 format because MySQL needs it later in such a format
- Install and setup basic MySQL
    - Generate root password and store it in .my.cnf file
    - Configure MySQL to “skip-name-resolve” for security reasons, other ways MySQL is trying to resolve connecting IP-s from reverse DNS and apply access control based on reverse lookup result if possible, and only fall back to IP if there is no reverse

DNS. We turned off this reverse lookup, to use deterministic way, so use every time only the raw IP address in access control.
    - MySQL is using Latin1 charset by default, we didn't change this to UTF8 because coTURN supports only 8 bit ASCII charset.
    - Setup MySQL Server and Client, Dump to use TLS key and certificate, and CAroot (we use the mentioned private key in PKCS1 format)
- Setup MySQL Replication.
    - Setup MySQL server identifiers "server-id" for each host, the master server has server-id=1 because replication is configured by default on slave side in MySQL to replicate from server-id=1 without any further configuration, so this is the practical reason we gaver server-id=1 to master server.
    - Enabling MySQL binary logging on master side.
    - Setup replication databases
    - Setup MySQL replication user on master side with limited access from client IPv4 and IPv6 addresses
    - Setup FERM open ports from client IP-s on master side to enable connect to MySQL master
    - Generate MySQL replication password, setup on master side
    - Setup encrypted replication using PKI.
    - Dump master database and import it
    - Start MySQL Replication on slave side
- Setup CoTURN
    - Install coTURN package from Jessie backport repository.
    - Import MySQL database Schema and Events on Master side
    - Generate MySQL password for coTURN on the Slave side and add new MySQL user with the right privileges and restricted to loopback IP addresses.
    - Setup options in coTURN config file.
    - Add coTURN to PKI group to access X.509 private key and certificate.
    - Configure ferm, /Netfilter, the linux packet filter/ to allow coTURN STUN/TURN service ports and the media relay port range.
- Apache
    - Install apache packages
    - Enable modules: rewrite, ssl, headers, expires
    - Disable default sites
    - Configure and enable the production site
        - Redirect from port 80 to 443, using the rewrite modul.
        - Configure SSL certificates and key files
- Download and setup GEOIP
    - Download MaxMind geoliteCity Database (IPv4 and IPv6).
- PHP5 setup
    - Install PHP5 and modules and php5-mod-apache2
- SimpleSAMLphp (SSP) Setup
    - Install SSP Debian package
    - Copy cert and configure PKI in SSP
    - Cron module
        - Generate random secret for cron module
        - Create /etc/cron.d/simplesamlphp configure with the generated user password
        - Enable and configure cron module
    - Configure metarefresh module

        - Use eduGAIN metadata
        - Enable metarefresh module.
        - Configure metadata directory ownership and permissions
        - Copy privacy statement (privacy.html)
        - Install and setup attributescope module
        - Configure apache2 SSP config
        - Increase PHP memory limit and max execution time limit.
- COMPOSER
    - Download and install composer
- WEB Applications
    - Checkout web application from git repository
    - Setup MySQL password
- Main Web application dependencies
    - "phpmailer/phpmailer": "^5.2" to send out emails
    - "hackzilla/password-generator": "^1.1" to generate password
- REST API dependencies
    - "slim/slim": "^2.6" The REST API is using slim php micro framework
    - "zircote/swagger-php": "^2.0" Swagger API is used to generate documentation from the php source of the API
    - "geoip/geoip": "~1.14", The originally built in PHP geolite implementation doesn't have support for IPv6, so I had to install this package to support IPv6
    - "mjaschen/phpgeo": "^0.3.0" REST API ordering result distance between client and server coordinates. We use Vincenty's Formula to calculate the distance between two coordinates.

3.6 Database Schemas and extensions

We use a MySQL database server and master-slave replication to distribute the shared secret. The two different types of credential methods have separate databases, and are replicated accordingly.

3.6.1 Database schema for Long Term Credential

We use the coTURN database structure with a slight extension, in order to identify the federated user. We thus added the fields “eppn”, “email” and “displayname” to the turnusers_lt table:

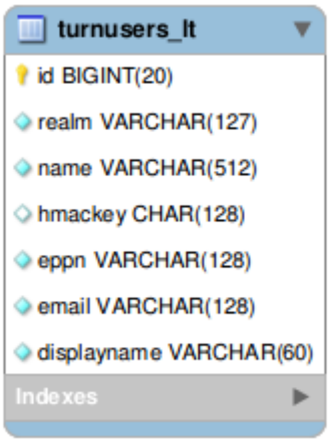

Figure 2: Table `turnusers_lt`

3.6.2 Database schema for REST support

The REST API database structure is also based on the default coTURN SQL schema, but was extended with more tables to store the coTURN server IP addresses and locations, plus the different service types that they provide.

In the database on the master side, there are events that generate the new shared secret key daily, removes the obsolete old key and deletes api_keys that are older than one year.

To add a new STUN/TURN node, a manual registration process is currently used. It uses the coTURN IP service table to store STUN/TURN server FQDNs, IPv4 and IPv6 addresses as well as a description of the STUN/TURN service, transport protocol (udp, tcp, sctp) and portnumber. We do not have a self-service or web interface for this function yet, but this could be implemented at a later stage to allow the server operator to manage his servers and its offerings.

About the tables and structures:

- The “token” table contains the `api_key` used for granting access to the REST API
  - The created field is the timestamp of the record insertion in the table.
  - The token could be requested after eduGAIN auth and could be revoked after a year. It is done by a MySQL event called clean token:

```
EVENT `clean_token` ON SCHEDULE EVERY 1 YEAR STARTS '2015-01-01 00:00:00' ON
COMPLETION NOT PRESERVE ENABLE DO BEGIN
DELETE FROM `token` where created + INTERVAL 1 YEAR > NOW();
END
```

- The “turn_secret” table is used to store the shared secret used for credential generation.
  - The `timestamp` field contains the date of the record insertion.
  - The `SharedSecret` event generates a secret and clean keys that are older with day or longer of the actual time:

```
EVENT `SharedSecret` ON SCHEDULE EVERY 1 DAY STARTS '2015-11-25 00:00:00' ON
COMPLETION NOT PRE
```

```
SERVE ENABLE DO BEGIN
    DECLARE secret VARCHAR(42);
    SELECT SUBSTR(CONCAT(MD5(RAND()),MD5(RAND())),1,64) INTO secret;
    INSERT INTO `turn_secret` (`realm`,`value`) VALUES
('lab.vvc.niif.hu',secret
);
    DELETE FROM `turn_secret` where realm='lab.vvc.niif.hu' and timestamp +
INTERVAL 1 DAY < NOW();
END
```

- The “server” table contains the FQDN of the STUN/TURN servers
  - It has a one-to-many relation with the “ip” table
- The “ip” table contains
  - The server ip address
  - GPS coordinates of the IP based on geolite database lookup
  - Preference value to help in ordering multihomed servers. e.g. it could be used to express IPv4/IPv6 preference.
- The “service” table contains the service port and protocol description
  - Preference value is to help in ordering of services.
  - Transport protcol udp, tcp, stcp etc.
  - uri_schema stun/turn
    - If secure communication used then stuns/turns

The database structure and relations are further explained by the Entity Relation Model diagram below:

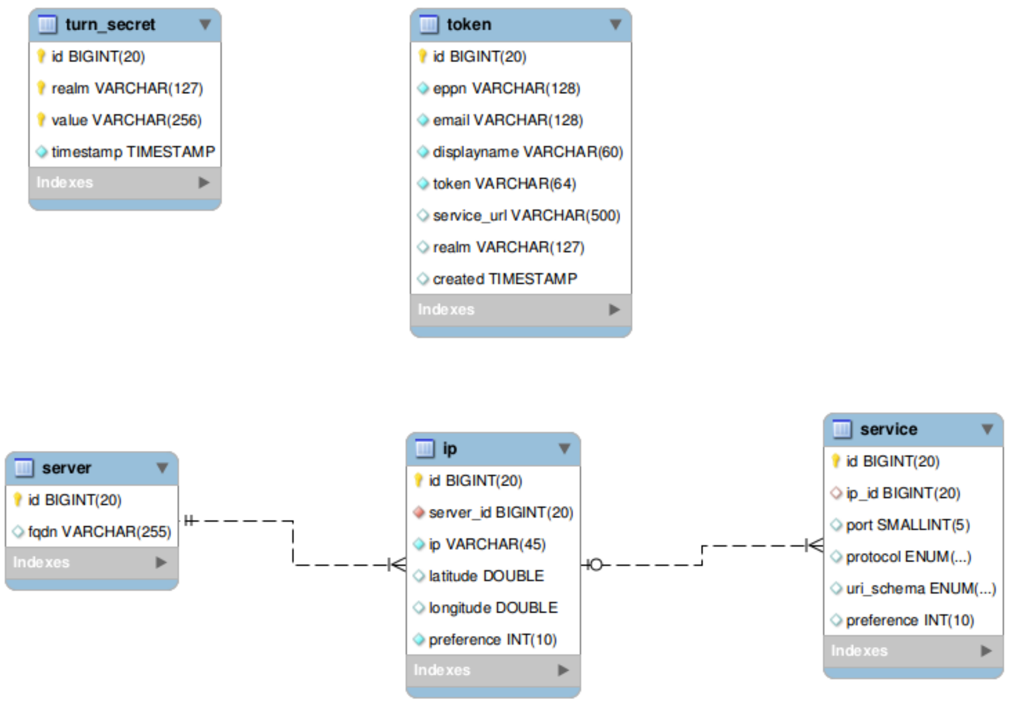

Figure 3: Database structure and relations

3.7 PoC Web Interface and Authentication schemas

The PoC service is currently available on https://brain.lab.vvc.niif.hu.

The Web interface source code is available on the (public) GitHub repository: https://github.com/misi/stun-web.

Access to the pilot service is limited to the NREN community and protected with eduGAIN AAI by propagation from the Hungarian Identity eduID Federation (HREF). After logging in, the user may request access credentials for the STUN/TURN infrastructure (i.e. username and password for the Long Term Credential Authentication Mechanism and a token for accessing the REST API). The REST API has a very simple Swagger-based documentation and test site here: https://brain.lab.vvc.niif.hu/restapi (temporarily the /stun and the /turn GET calls results are identical).

3.8 REST API

The REST API and the Swagger UI-based test/doc site source are available on the following (public) GitHub repository: https://github.com/misi/stun-api.

3.9 Geolocation for load-sharing and load-balancing

Using the closest possible STUN/TURN service yields the lowest round trip time, thus minimal delay and a better end-user experience. In theory, load balancing and load sharing may sound very simple, but implementing it in the service showed it to be complex. We go through the different supported authentication mechanisms and show how geo load-sharing and load-balancing was achieved.

3.9.1 Geolocation for Long Term Credential (LTC)

In our current testbed we utilize two Hungarian servers that provide the STUN/TURN service PoC with LTC authentication mechanism. Other nodes can be added.

We investigated the following mechanisms to provide load distribution and a way to find the closest server:

1. **Simple round robin DNS**

   Not desirable as it does not return the closest server to the client; only for load distribution.

2. **DNS geographic-based load balancing**

   This is what we are currently using. The Debian bind9 package has a built-in patch that provides GeoIP functionality. Based on that we may create an acl and create different views for different groups of countries. Partitioning is thus based on the country code, which is resolved from the IP of the DNS lookup client (resolver). It was configured according this guide: http://www.caraytech.com/geodns/.

3. **Anycast IP load-balancing**

   Anycast IP load-balancing has, like any other mechanism, its pros and cons. This setup needs a lot more work on the network side than the others. However, it works on the lowest layer in routing of the Internet Protocol and is very robust. To start a service like this, we have to request from RIPE, or a similar large IP registrar organization, a Service provider independent IPv4 and IPv6 address range, a /24 for ipv4 and /48 for IPv6. It then needs configuration, propagation in BGP and configuration in the local BGP filtering to be accepted. This is time consuming, and every European NREN would have to be asked to accept such prefixes.

3.9.2 Geolocation for the Time Limited Long Term Credential (REST)

The PoC service that facilitates the REST auth mechanism is provided by one Norwegian, one Portuguese, and two Hungarian nodes. The REST API requires some information from the user in order to be able to find the closest server to his location (e.g. IP address, a GPS coordinate or a Country Code). Fetching the IP address is the easiest to implement, as it may be extracted by the script (e.g. PHP) served by the web server. This is also the most convenient alternative for the REST client service and the WebRTC service operator. It may not be without implications, however, since it makes user tracking possible and could thus raise concerns about privacy.

In this PoC, as it is run “in-house” by the NREN community, we do not have any concerns over privacy issues. Our pilot REST API is therefore implemented using IP address; it receives the client IP address from the API call and returns the two closest servers based on GeoLite database GPS coordinates. It uses the Vincenty formula to calculate the distance between the GPS coordinates. Should the client IP be missing in the REST API, the API will return two random STUN/TURN servers.

To make the client service more secure, we could resolve to using the country code only. This would, however, require the client service to implement a mechanism to resolve the country code from the end user IP address. From a privacy point of view, it is nonetheless a better option and should be considered a mandatory requirement (for external service providers).

4 Lessons learned

A continuous service status monitoring, with active tests, is very important and we propose to place more emphasis on this in a possible later phase.

We also see the need for an operations team to look after the service monitoring, troubleshooting end-user issues, updating OS packages and continuously developing the service to keep up with the fast evolving standards and browser implementation changes. We also learned that browser STUN/TURN implementations differ a lot and that they are slow to take on board these standards.

We are very satisfied with the PoC results. They suggest to us that only small updates to the pilot are required in order to build a "real service".

4.1 Technology strength

We have already established that ICE/STUN/TURN is the only open standards-based technology available to eliminate the barriers imposed by firewalls/NATs. It realizes a connection between peers to make possible end-to-end real time communication and tests the connection in order to make the communication network more reliable. It also handles IPv4, IPv6 multi-homed situations and IPv6 transitioning.

In coTURN we found a simple, but rock solid and reliable implementation. Written in C, it is fully optimized and designed with a very efficient CPU and memory model. It uses libevent2 as a high-performance, industrial-strength, network IO engine. We think this implementation is ready and suitable for use in a real service.

4.2 Technology shortcomings

We do not see any standardization level problems. STUN/TURN standards are mature and well implemented.

Although we did find that the new OAuth access token credential mechanism standard is not yet implemented in browsers, it is nonetheless promising to find it in the roadmaps for the biggest browser vendors.

We found that there is a lack of PHP implementations of the Authenticated-Encryption with Associated-Data (AEAD) algorithm, but worked around this by using a coTURN client library to create the OAuth access token with AEAD.

One issue regarding the coTURN implementation is that it is not yet UTF-8 ready, but rather uses 8 bit ASCII encoding for strings. The implication of this is that UTF8 usernames cannot be stored or used. UTF-8 support is, however, on the project's roadmap.

Although we did not find any major problems with this technology we feel confident that any issues/shortcomings would be addressed within reasonable time; this is a field that evolves rapidly, with a number of significant companies depending on it.

WebRTC is a complex technology leveraging ICE/STUN/TURN. It continues to expand its market share, with a promise of continuity, reliability and quality. WebRTC has contributed significantly to make ICE to become a mainstream technology.

4.3 STUN/TURN service meets R&E community needs

Assumed real-time communication requirements from our community:

- User friendly UI and UX
- Instant and reliable communication from anywhere (place/network)
- Communication from any device (desktop, tablet, mobile)
- Communication between anyone
    - Global reach
    - Ease of finding contacts
    - If possible, standards-based
- Secure
    - Trusted
    - Use eduGAIN AAI if possible

The community demands a multimedia communication service that can be used anywhere and anytime, on any device with anyone, in any context. Our ICE implementation, coupled with the STUN/TURN servers, provides a proof-of-concept that demonstrates how this may be achieved. It tests connectivity before any media travels through a newly established connection, and continuously tries to detect failures and recover if possible. It helps to establish a tested and error free communication channel, regardless of network conditions and connections (local network, wireless, 4G/mobile), IP protocol version, or any number and combination of NATs and firewalls.

We can see WebRTC deployments growing extremely fast. And since existing platforms, such as VoIP, video conferencing and other standards-based communication systems (e.g. SIP and XMPP/JABBER) could also benefit, we propose to operate a STUN/TURN service for our NREN community.

As well as real-time media transmission (e.g. streaming and video conferencing), real-time data collection (e.g. WebRTC data channel) also depends on the ICE and STUN/TURN infrastructure. The WebRTC data channel opens new ways for real-time data collection from sensors, file sharing and even peer-to-peer content delivery networks. In other words, the technology has applications beyond audio and video media transmission and, as such, our community could benefit further from a distributed STUN/TURN service.

5 Conclusion

We note that ICE and STUN/TURN standard technologies are getting more and more attention, traction and adoption, and they are used more and more widely. Its adoption is already happening in the communication networks of our community. To build and operate such service locally for every NREN, however, is not feasible due to limited NREN resources.

We propose that a European, or even world-wide, co-operated NREN STUN/TURN service is worth serious consideration. An important aspect to study in this regard will be whether to build/make this service “in-house”, or buy from the market. This will, of course, require an in-depth cost and market analysis beyond the scope of this work.

Build or Buy?

In our observations, there are very few service providers that provide STUN/TURN services. This is likely due to the fact that most of the market players are building their own silos, running their base layer and STUN/TURN service in-house. When considering offers from service providers, we have to take in account - and pay extra attention to - the real-time data security and privacy concerns (e.g. to avoid interception during media relaying, avoid user tracking or any other form of data collection). We have to investigate what kind of authentication mechanisms the service providers provide, and how they solve the closest server location problem. In the case of TURN, it is especially important how the STUN/TURN servers are distributed around the globe (location and numbers) and what kind of mechanism(s) they use to choose the closest server.

Considerations for Future Directions

We propose to:

- Validate if it is possible to use container paradigm with the STUN/TURN pilot and investigate what benefits and/or drawbacks such a solution could introduce.
- Check and validate Anycast IP-based load balancing with Long Term Credential
- Extend our REST API features with nearest server location function based on the End User Country Code or GPS coordinates.
- Set up service monitoring:
    - Actively monitor with test binding and allocation requests on different kinds of protocols, using web browsers or the coTURN test client the STUN/TURN service.
    - Monitor hosts OS basic services
    - Monitor coTURN admin interface information
    - Monitor with the browsers’ peerconnection implementation. WebRTC Call tests.
- Set up a central log server and replicate all servers to log to this to help in more effective troubleshooting.

- Support OAuth authentication mechanism. We investigated and found lack of client implementations in two most used browsers. Firefox is not implemented, and does not exist in Chrome (https://bugs.chromium.org/p/webrtc/issues/detail?id=4907).
- Investigate extension of the service with STUN Origin support. And based on Origin the selection of REALM (https://tools.ietf.org/html/draft-ietf-tram-stun-origin-06). This is implemented in coTURN.
- Investigate extension of the service with bandwidth, connection limiting per user and/or global. (http://tools.ietf.org/html/draft-thomson-tram-turn-bandwidth-01). This is implemented in coTURN.

WebRTC and STUN/TURN technology deployment is already huge because it exists in almost every web browser, allowing an increasing number of applications to utilize this technology. WebRTC is not restricted to the web browser; it also has native, open source, implementations which allow native applications to the party (including mobile OSes such as Android and iOS).

We see that the "Internet of Things" and Smart City applications are already using the WebRTC data channel as the transport layer for real-time data collection from sensors (hence they are also using underlying ICE/STUN/TURN technologies). We expect to see a lot more of these types of applications in the future. These apps will also exploit all of the benefits of the ICE/STUN/TURN protocols.

With an exhausted IPv4 address pool, NAT and multiple layers of NAT will continue to make barriers. A smooth transition to IPv6 is still also a difficult task, but ICE/STUN/TURN provides a way to address these complex challenges. We don't see any other competing technology that could fulfill the Real Time Application NAT/firewall Traversal requirements like ICE do. We are, as such, confident that the usage of this technology will only continue to grow. It will also spawn various extensions, adding to the continuous improvement of its implementations.

In closing, attention must be drawn to how this technology may add value to our community. And, not least, what actions (immediate and future) are required from us in order to reap the most benefits both in the shorter- and longer term.

Based on our findings from the piloted ICE/STUN/TURN service; to provide and establish a STUN/TURN service for our Higher Education and Research community would not be a bad place to start…