PeerGaming – Share the Fun

Creating a Client-Side Multiplayer Gaming Framework for the Web, which handles Distributed Logic on Multiple Systems in “Real-Time”

Bachelor Thesis

Supervisors

Prof. Dr. Christin Schmidt (AI)

Prof. Dr. David Strippgen (IMI)

Stefan Dühring

April - July 2013

Hochschule für Technik und Wirtschaft Berlin (HTW)

International Media and Computing (IMI)

Student No.: s0531721

Abstract

Creating a multiplayer game or implementing a mode for various players can be quite challenging. In addition to the original client, developers need to take care of the backend as well and deal with additional hard- and software requirements. A persistent infrastructure has to be setup to provide a layer for communication and synchronization between the different systems.

PeerGaming is a client-side multiplayer gaming framework for the web, which helps you to solve some of these problems and connect multiple browsers directly using WebRTC.
It establishes connections, creates channels and provides a simple interface to ease the process of creating your own game.

As a game developer, you shouldn't be struggling with handling network issues anymore!

Dedication

In remembrance to all connections which don't exist.

Table of Contents

Table of Figures

If no additional information are provided, the graphics are created by the author and based on their according context.

1 Introduction

The following will provide some insight about the current situation of connectivity and introduce the problem I like to solve in the thesis.

1.1 Background

For the first time in history were more smartphones sold than feature phones in Q1 2013. According to a study grow the market about 43% compared to the previous year.[1]

There are several reasons for the ongoing interest – whether it be the basic idea of accessing the internet at any time via a mobile connection or using new features provided by the latest technologies. Undoubtedly one of the most important aspects are the mobile applications and additional functionalities that come along with them. Aside from direct communication is playing games one of the major use cases on these devices.

Traditional entertainment system, such as mobile handhelds like the Nintendo DS, the Playstation Portable or even home consoles like the XBOX, have therefore to compete with these phones and tables.

As handsets are replacing these specific systems more and more, the vendors are following the trend and extend their systems with missing features to improved their web capabilities. While the NDS was the first commercial successful console which got a browser as a default system application, the Nintendo WiiU even offers a Web Framework to create games and promotes them as first class citizen on their distribution channels.[2]

As developing software for different platforms with a specific SDK or programming language often requires more time and additional funding than a cross-browser web application, the interest in rich internet applications like browser based games is increasing.[3]

1 http://www.idc.com/getdoc.jsp?containerId=prUS24085413
2 https://wiiu-developers.nintendo.com/
3 http://readwrite.com/2012/06/05/5-ways-to-tell-which-programming-lanugages-are-most-popular

1.2 Problem

The best way to reach a large audience is releasing a product on multiple distribution channels. Referred to video games, this can lead to creating a multiplayer game in the browser – to attract a variety of players and keep the porting costs as low as possible.

After I did some research (see 1.3 Methodology), I found different approaches for the development of a game in a browser. Unfortunately none of these were simple or really appropriated towards the game design. Previous projects I worked on were mainly story driven and just for a single player, so I have limited knowledge about networking and hoped for a best practice guidance which allows immediate feedback via rapid-prototyping.

In the end I had to setup a network infrastructure, get a server and maintain the it's code in another programming language. During the development I recognized that their will be similar tasks required for each kind of real-time multiplayer game, disregarding its actual content.

To prevent repetition and help other developers who are perhaps struggling with a similar problem – I decided to create a client-side multiplayer gaming framework for the web, which handles distributed logic on multiple system in “real-time”.

1.3 Methodology

Without prior knowledge about multiplayer games in a browser or networks, I decided to pick a few examples and examine on which technology they are based and how they got created.

Smaller browser based games are often available on specific portals, while larger titles are available on their specific domain. Noteworthy games I found were Runescape[4], Canabalt[5], PolyCraft[6], BrowserQuest[7] and HexGL[8].

Although some of them are commercial licensed and others are open source, they provided a good overview about the environments and programming languages which can be used in a browser. It ranges from proprietary plugins up to standard web technologies.

The multiplayer aspect is also realized in different ways as well. While some are directly participating at the same game session, others use an indirect approach and strife for a new highscore. This can either be done synchronous (turn-based waiting for the input of others) or asynchronous in real-time without slowing down the actions.

A detailed analysis about the different tools for the creation can be found in “3. Analysis”.

4 http://www.runescape.com/
5 http://adamatomic.com/canabalt/
6 http://polycraftgame.com/
7 http://browserquest.mozilla.org/
8 http://hexgl.bkcore.com/

1.4 Structure

At first I will explain more about the current state and provide an overview about the essential topics: games, networks and their usage at multiplayer games in a browser. Afterwards I will present existing solutions and point out the difference towards my idea.

In the main part, I will summarize the basic concepts of the framework which influenced the actual implementation. This will be described by following the different steps which are done by users and developers. Aside from the usage, the core features and challenges of the development will afterwards be mentioned - just before the project will be evaluated at the end. With a reflection about the expectations, limitations and plans for upcoming versions I will complete the thesis.

You can find the source code of the framework on the enclosed CD, along with some additional material for the project. Since the whole projects is based on the idea of open knowledge and free culture, the latest version is also available online. Further details about the setup can be found in the attached guide.

2 Foundations

During the research I learned more about certain topics, which I like to explain in advance. It should provide a basic understanding of the terms and common usages of patterns. Specific abbreviations can be found in “8. Glossary”.

2.1 Games

There are various descriptions about “games” and “play”, but none of them can be used as an absolute definition. A reason are the perspectives, from which specific point of view the aspects are regarded. The different cultures also influenced the idea of games, for instance by promoting public events or reserving terms of the language.

Games can be divided in to different kind of categories. The groups are defined by similarities and reused patterns either by their medium, the actual content or the amount of players participating. As games can offer alternative modes – they can be found in multiple sections.

The most simple classification of a game is judged on the material: is it analog or digital. Many traditional analog boardgames were later on also ported as digital video games. Regarding to video games, they can also be classified by their specific platform they got released. Aside from consoles, computers or mobiles, browser defines nowadays their own category. Although their running on top of different kind of systems, games distributed over the internet and directly accessible are called browser based games.

Another filing can be done by the core mechanics used in a game. Independent from their physical or virtual existence, each game can focus on qualifications like luck, fitness, strategy, tactics or knowledge. Video games will also add sub-genres like Real Time Strategy, First Person Shooter or Role Playing Game to be more specific in terms of their gameplay and environment.

The level of difficulty and the targeted audience can also be a factor. Casual gamers prefer simplicity with a short introduction, while core gamers or professionals are more interested in the depth instead of the shallow experience.

Social components can also be a criteria: in opposite to a single user, games with multiple participants can be defined as multiplayer games. The interaction between those players and game elements can again be either competitive or cooperative.

The framework and thesis itself will focus on the technical aspects of a multiplayer game, which can be any kind of genre, setting or style.

2.2 Networks

Computer networks are the backbone of our modern society, as they allow accessing data from remote systems and communication in real-time with other people.

Although true “real-time” barely exist from a technical point of view – its often used to describe a signal transport without noticeable delay. A high bandwidth connection and low network latency will create such perception.

2.2.1 Architectures

Networks use digital or physical transports to connect multiple systems with each other.
The infrastructure can be defined by different criteria, for instance the instruction types and data-streams internally used. A more common approach is the classification by the communication roles of the systems in the application network.

On the one hand is a client, which opens a socket to create a connection to another system.
The machine which listens to this incoming connections is called a server. Their relation and functionalities keeps the same during the runtime. A multi-tier design uses the client-server architecture with a strict hierarchy by declaring the roles for each layer independent.

In contrast to the classical client-server model, a system which is able to change its role or act as both can be described as a peer. It can create connections and listen to them – regarding their current state and implementation even at the same time.

Both kind of systems are used nowadays and depending on the features of the application, specifically designed with the benefits and disadvantages in mind. Reliability, accessibility, maintainability, security, latency and costs are a few factors for measurement.

While the client-server architecture provides a central host for transferring and storing data, its also a single point of failure in the infrastructure. The network is limited by the resources the computer provides. Even if a switch is used to balance the traffic by addressing multiple servers instead of one, there is a restriction on scalability how many concurrent clients can be connected and handled.

Peer-to-peer system have to take care of the opposite: as no central location is available to send or retrieve information, the peers don't know which other peers are online and available. The initial connection is more complex and requires heuristic data or an additional service for bootstrapping. Without a permanent host persistent data can also not reliable stored on a single system.

2.2.2 Topologies

Aside of the roles or technical components - a network can use different kind of structures to organize the communication inside. The connections of the nodes (server, client or peer) defines the flow and efficiency in handling data. Similar to architectures, depends the topology on the specific needs of the application. As peer-to-peer systems are more flexible in their organization, the design is significant for the result. Common topologies are: *

Fully Connected Network

Every peer is connected with each other peer in the network.

+ direct connections / short distance without a proxy
- doesn't scale well (resource management of 1:n connections)

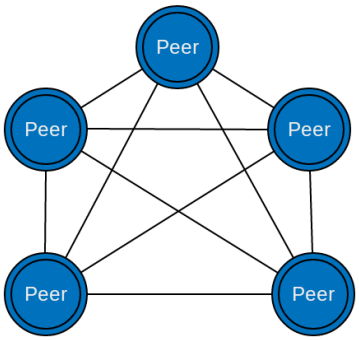


Figure 1: Full Connected Network

* *https://github.com/Raynos/topology , 23.07.2013 - 11am*

Star Network

Every peer is connected to one peer, which manages the data communication as a central hub.

+ peers just handle 1:1 connection, simple to add new nodes
- dependency on the central peer (see client-server)

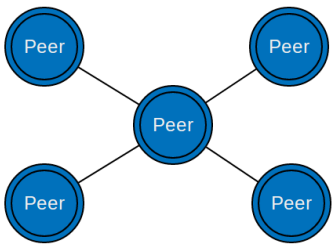


Figure 2: Star Network

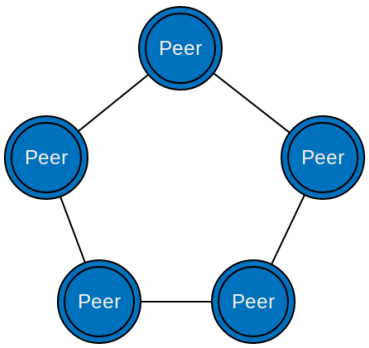


Figure 3: Ring Network

Ring Network

Each peer is connected with two others, defining a closed loop.

+ always 1:2 connections
- multiple points-of failure, slow transfer over long distances

Tree Network

Each peer takes a position in a hierarchy and is either a root node with childs or a “leaf” of one.

+ support subnets on disruption
- requires a defined hierarchy matching

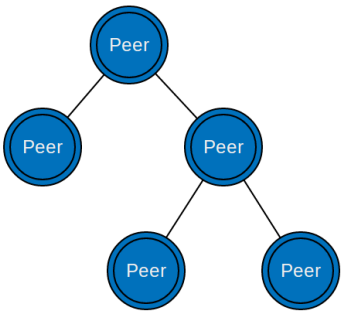


Figure 4: Tree Network

2.3 Web Technologies

Nowadays is the internet the largest global network world wide – which connects many smaller networks and systems. The “web” describes a system defined by links between different documents and provide an accessible infrastructure for various services on top.

2.3.1 Browser

A browser is an application which enables the user to visit websites by following the route of referred documents. While previous version were purely based on text and ASCII encoding, modern variants include even audio and video support.

There are currently 5 browser vendors at the time which cover the majority of internet users: Mozilla Firefox, Google Chrome, Apple Safari, Microsoft Internet Explorer, Operas Opera. With the growing interest in the mobile market, they also provide mobile variants of their browsers on the specific platforms along the desktop.

Compared to other platforms the web suffers especially from outdated software. Developers just create a part of the experience, since the browser is the real client which renders the site or executes the logic of an application. The capabilities and usable features are therefore defined by the systems the visitors are running.

So even though its easier to access web content, compared to a “native” application on a device where an additional installation is necessary for the start, it's limited by the sandbox its running in. The inconsistency of the different browsers implementations, as well as unknown hardware requirements needs to be considered by web developers.

2.3.2 Standards

In addition to the general infrastructure, a common aspect of the web as a platform are the different standards – on which users, developers and vendors can rely on. These include recommendations by organizations like the W3C, the IETF and Ecma International.[9]

Although it was originally just designed for simple documents, the web nowadays consists of different components including visual elements, multimedia streams, dynamic text content and interactive applications. Standard web technologies include HTML for the content, XML as a structure and CSS for the styling.

During the last decade, the idea of running real applications directly in the browser got promoted. Shifting heavy computation towards the client and extending the possibilities, allows to create thick clients and leverage the server resources.

9 http://www.webstandards.org/learn/faq/#p2

2.3.3 JavaScript

JavaScript is a dynamically typed, object based, interpreted programming language. It was originally created to run in browsers to extend their functionality, but can also be used in different environments which are using scripts.

The current version of the language is 1.8.5 and based on the ECMAScript 5 specifications from 2011 defined in ECMA-262[10]. Its written and proposed by the Technical Committee 39 (TC39), including members from the major browser vendors and technical experts.

Although it doesn't belong to the official W3C standards, JavaScript is as a part of the standard web technologies. While HTML is static and represents the content, JavaScript can be used in conjunction with the DOM to add behavior and interactions. Creating new Elements, adding inline styles, changing the text or requesting information is possible by using the API provided by the browsers.

The language itself got created in 10 days[11] and because of the dependency on a browser, it had to be defined with backwards compatibility for older versions in mind. Therefore it uses prototypical inheritance for extensibility.

It also promotes functions as first class citizens, which allows to simply adapt towards the nature of functional programming and write expressive code. In addition it got short literals for objects, arrays, numbers, strings and regular expressions.[12]

The term “HTML5” is frequently used as a buzzword for modern web technologies, which can be used with JavaScript. Despite its origin as the fifth revision of the HTML standard, new browser APIs and CSS3 properties are commonly referenced as well.

10 http://www.ecma-international.org/publications/standards/Ecma-262.htm
11 http://www.computer.org/csdl/mags/co/2012/02/mco2012020007.html
12 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Values,_variables,_and_literals#Literals

2.4 WebRTC

WebRTC is the shorthand for “Web Real-Time Communication”, a set of new JavaScript APIs for the browser – which allow real time communication. It is defined by groups of the W3C (WebRTC) and the IETF (RTCWeb).

Google acquired Global IP Solution (GIPS) in May 2010 and obtained with them technologies for real-time voice and video encoding. They provided them royalty free and proposed a JavaScript API to use them directly in the browser. Together with Mozilla, Opera and guidance by CISCO and Ericsson – they promoted the draft. Meanwhile Microsoft works on its own specification CU-RTC-WEB, regarding a different implementation of the used technologies.[13]

2.4.1 API

Currently the API is still in the editorial process and yet not a full recommendation for the browsers. The WebRTC specification[14] consists mainly of 3 components:

- **PeerConnections** can establish a direct connection between two clients
- **MediaStreams**[15] allows to retrieve continuous signals from input sensors
- **DataChannels** are able to send and receive arbitrary data across a peerconnection

As the APIs to access these parts are not final and still changing, not all browser support every feature. The best coverage have MediaStreams, which can be accessed by a prefixed version of “navigator.getUserMedia()”.

13 http://html5labs.interoperabilitybridges.com/prototypes/cu-rtc-web/cu-rtc-web/info
14 http://dev.w3.org/2011/webrtc/editor/webrtc.html
15 http://dev.w3.org/2011/webrtc/editor/getusermedia.html

A permission request will prompt to the users and after acceptance, provide a *mediastream* object which can either be used on the local system or send to a remote computer by attaching it to a PeerConnection. Aside from the majority of desktop browsers, even some mobile variants support camera records.

The implementation of PeerConnections and DataChannels on the other hand just recently shipped into the stable versions of the browsers. Currently only Firefox 23+ and Chrome 27+ support DataChannels and because of their different implementations, communication using DataChannels for text based messages is unfortunately not interoperable between them.

The setup and underlying protocols for using PeerConnections are described in the following.

2.4.2 Protocols

Different kind of protocols can be used as the transport layer to transfer data over a network. The most well know protocols for the internet are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

TCP creates a reliable connection between two systems. It automatically split up large messages, streams these data in chunks and re-assembles them on the other side to the original. Through the process it takes care of the package orders and guarantees the arrival.

On the other hand is UDP purely based on packets. It sends messages to a multicast group, on which every participant will receive the data. In contrast to TCP it doesn't provide extensive features like the conversion in smaller segments and the flow control to ensure the order. The error correction by the streaming interface is also missing – so part of the input can be corrupted or even lost during the transport.

While TCP is commonly used for 1:1 requests like in a browser to ensure the result, UDP transfers got the advantage of having a low latency and therefore are commonly used

at real-time communication. Since UDP doesn't provide an interface which handles the data, there is no additional overhead compared to a TCP stream. Therefore application which are updated frequently like media streams and games (were the loss of the previous package is not critical considering the fast delivery of the next frame) prefer the usage of UDP.

PeerConnections are using UDP for the transport and different application layers on top. By default the basic Real-time Transport Protocol (RTP) is used for handling messages. As a MediaStream will be attached to a PeerConnection, the signals will be encrypted and use the Secure Real-Time Transport Protocol (SRTP).

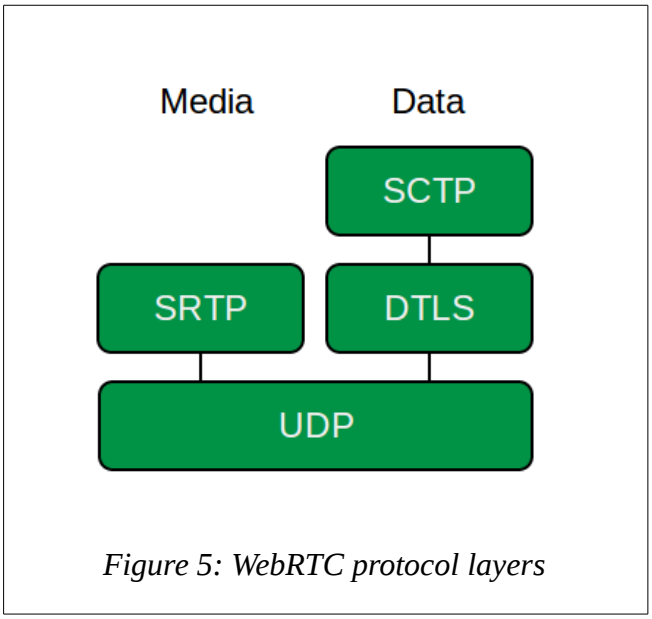


Figure 5: WebRTC protocol layers

Data send via a DataChannel will use the Datagram Transport Layer Security (DTLS). An upcoming version will also include the opportunity to enable a reliable DataChannel based on the Stream Control Transmission Protocol (SCTP), which provides a similar congestion control like TCP – but will be run over UDP.

2.4.3 Connection

WebRTC is implemented by following the JavaScript Session Establishment Protocol (JSEP), which defines an interface to control the Interactive Connectivity Establishment (ICE) framework from within the browser using a JavaScript API. It allows to create a direct connection between two systems which are behind network address translators (NATs).

The problem is that most computers will only know about their private IP on the internal network, but not their public address. In order to contact another system over the internet directly, it needs to obtain the public address of the partner.

The first step of the ICE implementation in the browsers is to specify the address of a server, which provides support for Session Traversal Utilities for NAT (STUN). A server using the STUN protocol will simply send the public IP back to the client, so it knows about its own address on the internet. In some cases can the public address not be resolved, as fore instance a strict firewall prevents accessing the information. Even though this happens to just 1 system out of 7[16], a fallback needs to be provided to still work properly. Therefore the URL of a Traversal Using Relays around NAT (TURN) server should added as well. If the previous request towards a STUN server fails to return the address, it will connect to a TURN server which will then be used as a relay for all transmitted data over the connection.

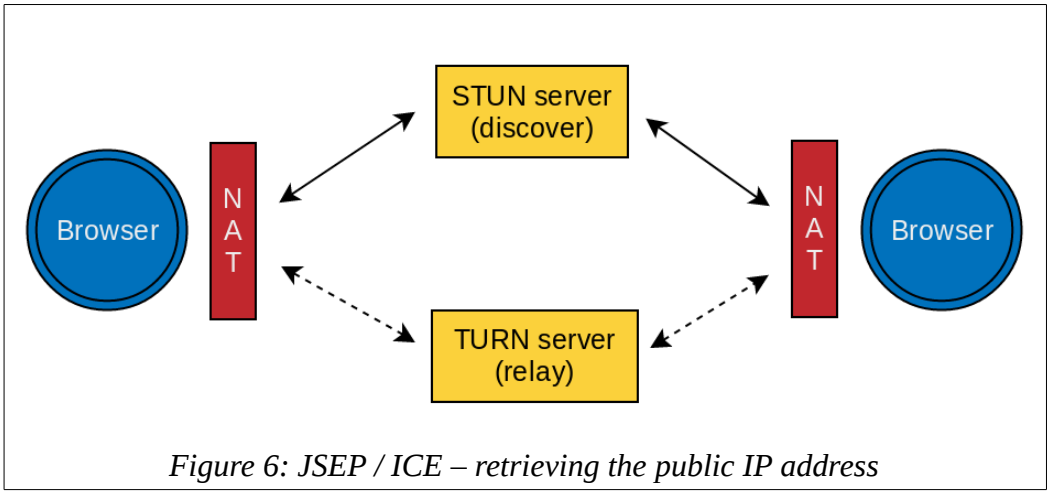


Figure 6: JSEP / ICE – retrieving the public IP address

16 http://io13webrtc.appspot.com/#51

Although this won't be a direct exchange of data anymore, it still works with the same interface on all cases.

After the public address is received by a STUN server or a TURN server sent a confirmation for the relaying setup, developers on the client side have to take care of triggering the next steps of the ICE framework. To stream media over the direct connection, both systems needs to be aware of the actual types and data which will be transmitted. The Session Description Protocol (SDP) is used as format to exchange these information. An SDP package contains not just details about the audio and video codecs, but also specifications for their resolution and restriction on bandwidth usage.

The ICE framework defines the establishment of a connection by an “offer-answer” scheme using SDP and address candidates. These candidates are a combination of network interfaces and ports, which will be used for hole punching.

One peer will initilize the process by defining a new PeerConnection and create an “offer”, which is an SDP package with optional constraints towards the media usage. He will then set it as his LocalDescription, which will invoke the gathering of ICE candidates by the framework. Both credentials, the SDP package and the candidates, needs now to be set on the remote system of the partner. The communication channel for exchanging the data is not specified by a protocol and can be anything from another socket connection towards an E-Mail.

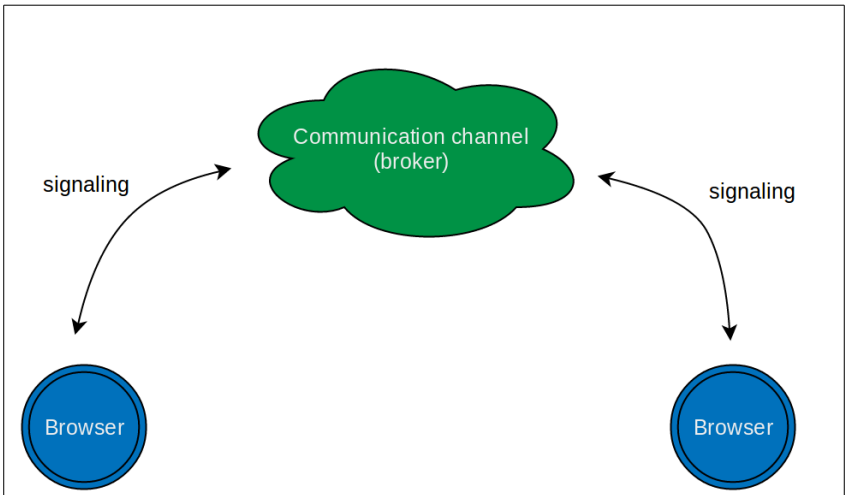


Figure 7: JSEP / ICE - exchange candidates & SDP container

As the signals are transferred to the partner – a new PeerConnection will be created, which will be the endpoint for the first peer. The ICE candidates will be added to the connection and the “offer” will be set as its RemoteDescription. Similar to the “offer” will then an “answer” SDP package be created, which will be set as its LocalDescription. The ICE framework starts to gather new candidates, which has to be send along with the “answer” to the first peer.

After the session description and the candidates are received by the first peer, the candidates will be added to the connection and the “answer” will be set as its RemoteDescription. The PeerConnection will then be established through an internally matching of the provided credentials. If a participant couldn't retrieve its public IP, the TURN server will be used as a relay for the communication instead.

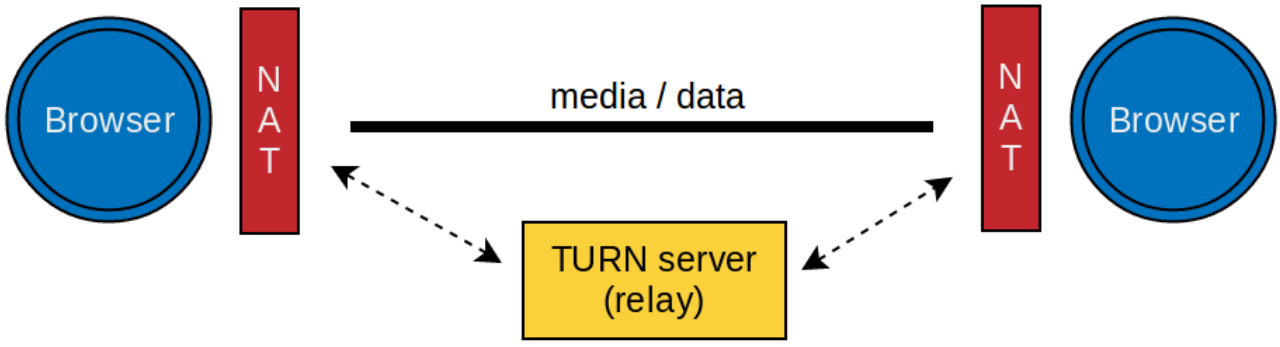


Figure 8: JSEP / ICE - established a connection to another peer

Depending on the usage and the initial situation, a re-negotiation with a new “offer-answer” exchange can be necessary. Every change towards the current state of a PeerConnection, like adding a DataChannel or attaching a new MediaStream, requires an update of the connection.

3 Analysis

After looking at existing projects, I point out my idea and the requirements for a real-time multiplayer gaming framework.

3.1 Market

There are several solutions available to create real-time multiplayer games in the browsers, each with their own advantages and caveats. Aside from coding everything from the scratch or relying on an external service, I further looked at complete frameworks which helps developers with the network communication.

3.1.1 Plugins

JavaScript is not the only programming language which runs in the browser, at least if you include plugins which provide their own sandbox and APIs. Using Silverlight Extensions, Java Applets, Flash Container or the Unity Webplayer[17] provide an alternative to create browser based games with multiplayer options.

All of them allow developers to use extended features with a consistent API disregarding the original browser environment, which could be outdated or limited by the vendor. The code for the game can now be written in Java, ActionScript or C# - which are not just statically typed, but also got import functions and promote object orientated programming with a classical inheritance. These will lead in general to a better structured code. In addition the projects are pre-compiled and can be obfuscated. Manipulations by users will be prevented, as direct changes during the runtime are not possible.

17 http://unity3d.com/webplayer/

As long as the according plugins are installed it works, but thats the restriction which overweights the mentioned benefits. None plugin wrapper is available for all systems or browsers, especially Linux systems are neglected by the companies. Furthermore mobile devices won't support any of these technologies for their browsers and Apple official declined any interest in Flash.[18]

Even if a specific plugin exists for the browser and is installed, the default settings in modern browsers prevent an immediate execution of an application. Firefox fore instance blocks major 3rd party plugins to spare resources, which improves the performance and security.[19]

Compared to a standalone executable version outside of the browser, the embedded applications are also less efficient running in their virtual machine.

18 http://www.apple.com/hotnews/thoughts-on-flash/
19 https://blog.mozilla.org/security/2013/01/29/putting-users-in-control-of-plugins/

3.1.2 Services

A common approach for creating real-time multiplayer games in the browser is using WebSockets. These are based on HTTP and provide a persistent bi-directional connections to a server, which allow developers to send text messages and arbitrary data. The server will then delegate the information by broadcasting it to all connected clients.

In contrast to the former technique of using polling with AJAX, there is the advantage of the event based model which just listen to changes instead of continuous sending requests. Moreover the connection stays open and can be reused, so the overhead by the HTTP header which we had otherwise will be reduced. As both sides are aware about the state of the connection, the server can also determine a disconnected client and close the session.

The implementation on the server side on the other hand is more complex than parsing a simple POST or GET request. Regarding the used technology, there are probably libraries available which provides an implementation and abstracted interface on top.

Aside looking for a library or even writing code at the back-end at all, there are also companies which provide a partial setup for the server side. Moreover they often got an SDK for their API to access the connection handlers directly from the client. Pusher[20] and PubNub[21] are just two services which offer this functionality for real-time data distribution.

In addition to handling direct messaging via WebSockets, Firebase[22] is a service which takes cares of persistent data as well. Developers can configure a remote database (a MongoDB), which is usable directly from the browser to store any kind information as binary JSON.

20 http://pusher.com/
21 http://pubnub.com/
22 https://firebase.com/

The advantage of using these platforms as a service (PaaS) is the reduced technical stack, which otherwise had to be handled directly. Through “outsourcing” the back-end and not writing code in different environment, allows to focus on implementation of the client part.
It can also be setup straight forward, without requiring any previous knowledge about the underlying technologies.

On the other hand is none of these services specified for handling games. They are basically synchronizing message streams between browsers, which have to be parsed on the client side to trigger a remote procedure call (RPC) for the intended action. As these platforms are running a permanent server for the connections, which has to be maintained and kept available, they cost money and increase their fee by the amount of transferred data. Although a free tier is offered for getting an impression – it doesn't take much till it scales and costs additional money.

3.1.3 Frameworks

As I looked at complete game engines with multiplayer support like Turbulenz[23], it was clearly superior to my expectations of a simple library. It includes many features like asset loading, physical simulations, graphic rendering and multiplayer session handling. Although the code itself is modular, it's quite complex and learning the usage of the API takes some time. Other engines like Playcraft[24] seems promising, but aren't public available yet and will presumably be commercial.

Therefore I searched for more specific frameworks, which claimed a focus on multiplayer gaming in the web.

One of them is the *HTML5 Multiplayer Kit*[25], which is a customized bundle of server side components and a wrapper for the client. On the back-end it uses redis for persistence and streams the data via Socket.IO. Unfortunately there is no test suite or public API to see how it actually works and as a commercial product it's not accessible by everyone.

Another framework is called *Pomelo*[26], a module for NodeJS to create scalable game servers. It doesn't force developers to use an additional client side wrapper, but provides hooks for Socket.IO to communicate with the back-end.

In contrast to the other projects, the Chilly[27] framework uses AJAX instead of WebSockets to exchange information with the server. As a result will the delay between sending requests and receiving responses much higher.

23 http://biz.turbulenz.com/developers
24 http://www.playcraftlabs.com/
25 http://eerie-studios.com/html5kit/
26 http://pomelo.netease.com/
27 http://chillyframework.com/

All of these frameworks are using a client-server architecture, which consist basically of a client side wrapper and a server side module for handling the communication. To use them, developers first need to setup an own server with their custom logic and delegate RPC calls towards the browser. The extensive usage of Socket.IO to send messages is also doubtful. Despite its popularity, does the WebSocket wrapper add features which aren't appropriated for low latency gaming. Each message will contain additional information to be identified by its custom channels and serialize the actual data as strings. This overhead can't be disabled and isn't necessary for most games.

3.2 Vision

As none of the existing solutions provides the support and features to get widely adopted, I wanted to create such framework by myself. It should be based on Web Standard Technologies and provide simplicity without a cumbersome setup.

To avoid additional costs and hardware requirements – it should use a P2P approach with predefined configurations. Since no server has to be maintained for handling the logic, the developer can focus on the game and write purely in the client side language.

Although the previous frameworks were specifically designed to create multiplayer games, all of them are missing the support for direct communication aside of text messages. Since the social aspect is important and should be included along the actual game content, advanced streaming capabilities for interactive media like audio and video should be offered as well.

Finally it has to be available for everyone: besides not depending on a proprietary technology or external plugin, it should further promote the open source culture through using a minimum restricting license like the MIT. Interested people will then be able to contribute to the project, helping to maintain and improving the code.

3.3 Requirements

To measure a difference towards the analyzed projects, the framework should fulfill the following assumptions in terms of usability:

1.) *No additional component has to be required.* The code itself should not rely on unreasonable installations by the user or script includes of the developer.

2.) *A server shouldn't be necessary.* To lower the costs and ease the setup, running an own server shouldn't be mandatory to create and deploy a game.

3.) *The majority of desktop users has to be supported.* A game created by a developer should be available to a large audience and attract many customers.

4.) *It should be usable with any kind of rendering*. By focusing just on the networking, it will be as little intrusive as possible and compatible with different game engines.

5.) *Plain data can be exchanged without defining remote procedure calls*. Sharing the state of a player is a common task which can be done automatically.

4 Design

In this section I will got through my ideas and explain some preliminary considerations I've taken before starting with the implementation.

4.1 Principles

The concept of the framework is based on three main aspects. It's always considered to be:

- standalone

  In contrast to other multiplayer modules for the browser - the framework tries to be agnostic and without further dependencies. Developers & gamers shouldn't require an additional server setup or a complex installation to use the script.

- configurable

  Many constraints are optional and customizable. As the developers are the ones who will use the framework - they probably know best which settings are appropriated for their situation and make their own decisions.

- simple

  Network or security issues shouldn't concern a developer who just likes to create a game. Aside from the default options, the whole API is designed for minimal input and graceful handling of parameters. It should work out of the Box!

Although performance, scalability and reliability are important as well – the developer experience should be more crucial for the design. Optimization often relies on the specific game implementation, so the framework should be flexible and just focus on the basic network handling.

4.2 Workflow

To determine the explicit structure and functions of the framework, it needs to be considered what kind of software the developers are creating and how they are going to use it.

A good developer experience depends not only on an extensive documentation, but more importantly on a lean interface with a concise API to solve the specific problem it's targeting.

4.2.1 User

Even though there are different kind of browser based games, the procedure by joining a multiplayer game is in general the same for all platforms. With a few additions, this will be similar to the traditional flow:

At first the user visits the site where the game will be setup. Then he creates an account to identify himself towards others and the game. After the login there is often a lobby - with the opportunity to contact other players outside a running session, a session represents in this case an active game with a defined rule set. This can be either done via a direct message or in a public chat. To start the game, the user has to confirm his intention by clicking on a button or pressing a shortcut on the keyboard. Afterwards he gets matched with others. It then differs on the actual game implementation.

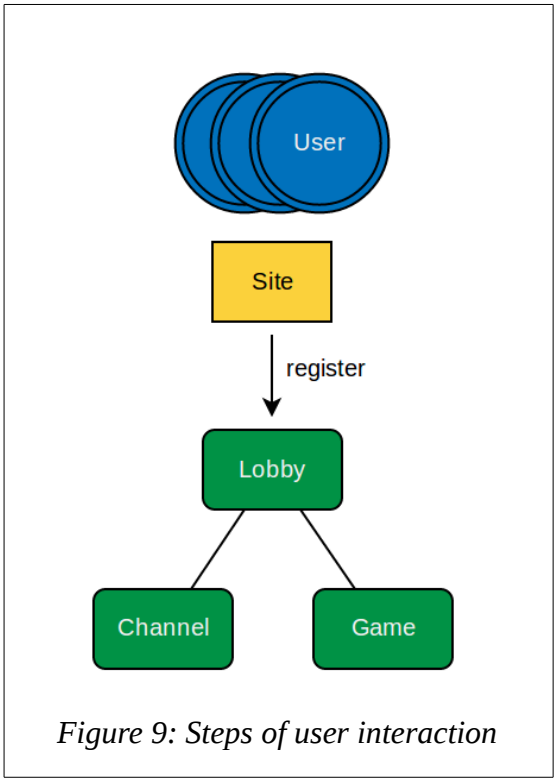


Figure 9: Steps of user interaction

A secondary channel to arrange a team could follow or just lead straight into a level. Sooner or later the game world will be created and a dashboard is commonly shown to inform about the connectivity and rendering progress of each player.

The game itself starts as everyone is ready to participate. After the game ends, the scene will change again and the players will be re-directed back to the lobby.

4.2.2 Developer

According to the users actions, developers have to take care of implementing these steps for each game. So the following tasks should be accessible by the framework to prevent repetition and ease the bootstrapping process:

- register a player | authentication
- create a lobby | separation
- send messages | communication
- change scenes | navigation
- start a game | invocation

In addition to these preparations, the developer will create the playable part as well. Therefore he needs hooks to receive data and functions to share own information with others. Despite the content or genre there are mainly 3 types of data involved, which got different permissions for accessing and changing their values.

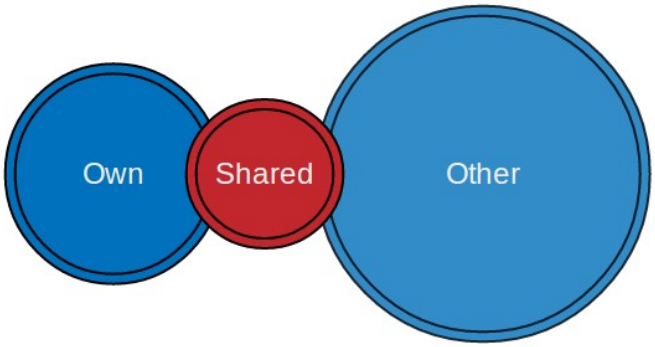


Figure 10: Data types in multiplayer games

own : specific information which are different for each player and should only be created / updated by the player itself (readable + writable)

other: a representation of the data by other players which are only used for rendering and shouldn't be manipulable (readable)

shared: information which can be accessed by all user like environment settings, NPC interactions and mutual computations (readable + writable)

Keeping these information synchronized between all players is often done by developers, although this could also be handled automatically by a framework.

4.3 Network Topology

The efficiency of a peer-to-peer network depends on the underlying structure it's based on.

I decided to use a full-connected mesh in which every peer is connected with each other. Using WebRTC (see 5.1 WebRTC) for exchanging information, it's currently not possible to create a InputStream for audio/video and relay the signal via another system. A direct transmission from the origin is required to replay the media.

Moreover, the main advantage of this approach towards others is the consistent amount of jumps to reach any target: 1. This will in most cases reduce the latency to a minimum.

On the other hand, the scalability will be limited by the resources of the “weakest” member and suffer by complexity. Even though this drawback had to be considered - it's minor compared to the advantages of an overall lower latency in most cases.

The idea of PeerGaming is to assist the creation of more multiplayer games. For this purpose, the framework has to be flexible for various genres and scenarios. As it can't solve every problem at the same time, the focus will be on small matches with up to 10 players by default.

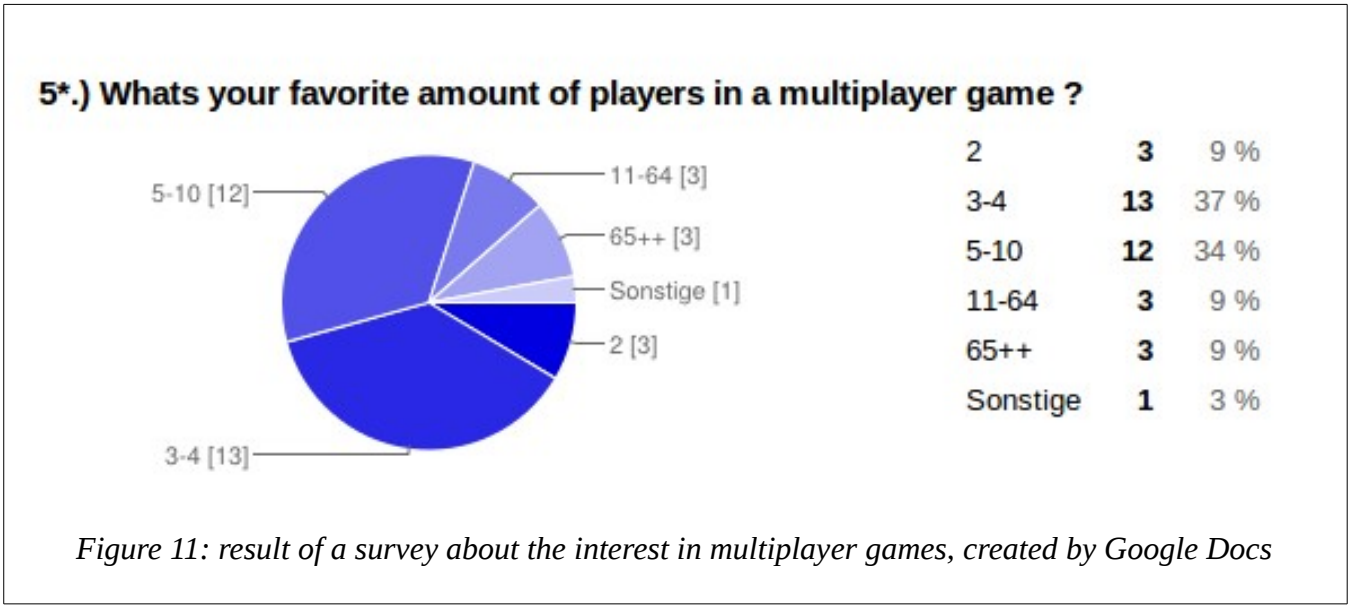


Figure 11: result of a survey about the interest in multiplayer games, created by Google Docs

4.4 Design Patterns

As the network interaction is handled asynchronous - sharing data or invoking callback functions can be complicated. Therefore I introduced a Mediator to ease the communication between different modules. Each corresponding instance will inherit the prototype from "Emitter", which provides them with a generic publish-subscribe functionality.
Another useful abstraction is the encapsulation of the internals by using a closure. Only a restricted set will be accessible as a public API for the developers, while the main logic is hidden behind a facade.

```
/** API Overview **/

var pg = {

  noConflict  : fn   - reset namespace
  VERSION     : obj  - refers to the current version
  info        : obj  - information about the state
  config      : fn   - configuration for the network
  watch       : fn   - notifications by internal events
  login       : fn   - set identifier and create player
  player      : obj  - own user instance (writeable)
  peers       : obj  - list of connected players (readable)
  data        : arr  - shortcut to access peers.data + player.data
  sync        : obj  - synchronized shared object
  loop        : fn   - synchronized rendering process
  channel     : fn   - handler for a "Channel"
  game        : fn   - handler for a "Game"
  routes      : fn   - define default and custom routes
};
```

5 Implementation

In the following I will explain more about the technical aspects of the framework, how it actually works and which specific features are realized.

5.1 Server

The framework uses WebRTC to create direct connections between the users on a site.

To handle the signaling process for the initial exchange of the credentials, an additional *PeerGaming-Server* is provided. Although it's not mandatory to use it – it offers a predefined setup to get started and takes care of common tasks. Otherwise developers have to solve them manually by choosing another communication channel.

In particular the server is used to:

- inform the visitors on the site about each other, while keeping a list of the users which are currently online for further upcoming connections

- create subnets for the current address, allowing users to be grouped at different rooms on the same domain (see 5.3.1 Rooms)

- automatically match peers, as the connecting process on the latest client will be triggered while there are at least two visitors on the same subnet

- relay the credentials, as transferring the ICE candidates and SDP packages towards their partner

Later it will also be used as a proxy to support basic OAuth using an external service to provide a login option with persistent data. An interface to access network or gaming data in real-time via a simple REST-API will be added as well.

The current implementation is done in JavaScript using the NodeJS framework. It supports Server-Sent Events and WebSockets for a permanent bi-directional connection from a peer to the server. The basic handling of the EventSource-Stream is done by hand, while for the WebSocket connection it uses the “websocket-node” library by Worlitze.[28]

28 https://github.com/Worlize/WebSocket-Node

5.2 Client

The framework consist of a single script file for the browser – which has a size of about 25kb.

As JavaScript itself doesn't provide a native solution to import other packages yet (a new method to include modules will be available in ECMAScript 6 NEXT [29]), I defined a separate file for each logical component of the framework and grouped them into directories. At the end of a build process will each of these partials be merged into one file and the Google Closure compiler minifies the code to reduce the overhead.

PeerGaming allows dynamic script loading as well, since the framework is using an universal module wrapper to support the AMD and CommonJS module definitions.
A developer can therefore inject a script tag after the initial rendering of the website, by defining the dependencies and including a script loader like RequireJS[30].

To reduce the pollution of the global namespace, the complete API of the framework is accessible via the object 'pg'. As former elements could already use the same reference, a “.noConflict()” option is available to restore the previous value.

One of the most important aspects of creating the framework is simplicity. Although the developers should always be in charge of decisions, the default configurations will already cover many cases to ease the setup. Therefore it will use a predefined address of a public available instance of the brokering server, leverage the NAT hole punching by using a STUN server maintained by Google[31] and fallback to a TURN server hosted by Viagenie[32].

29 http://wiki.ecmascript.org/doku.php?id=harmony:modules
30 http://requirejs.org/docs/why.html
31 https://code.google.com/p/natvpn/source/browse/trunk/stun_server_list
32 http://numb.viagenie.ca/

The developers can overwrite these settings with their own configurations calling ".config(custom)". If they even prefer to handle the matching and credential exchange at all, there is a ".noServer()" option to prevent the defaults. It expects a callback function, which will be executed every time an ICE candidate or SDP package is generated. With this approach the developer can choose it's own messenger independent from the actual game content.

A notifier for internal messages is available at "pg.watch" and can be used to handle errors.

5.2.1 Rooms

The URL will be matched to a path for a channel or a game.

As multiplayer games often differ between a lobby for social interactions and the game itself, I introduced the concept of a *room*. In general a *room* is a subnet which separates different groups and offers specific features according to its type. There are two kind of *rooms* - channels and games. While a channel just provides the basic functions to communicate, a game extends its possibilities. A developer can provide a bootstrapping function to start the actual rendering and define hooks which will be triggered as new users “enter” or “leave”.

The URL in the browser defines the room in which a player currently stays. As the user navigates and the path changes, the *hashchange*[33] event will be triggered by the window. PeerGaming parses the address and tries to match up the location. By default it will first look for a game id and then for a channel name.

Default Scheme:

channel → 1 parameter || http://{HOST}.{TLD/#!/{channel}/

game → 2 parameter || http://{HOST}.{TLD/#!/{channel}/{id}/

33 https://developer.mozilla.org/en-US/docs/Web/Reference/Events/hashchange

Developers can also attach handlers for a type in general or specify actions for certain rooms. Following events exist and will be dispatched:

Channels (inclusive Games)

“enter”	:	as an user is connected with all other peers in the room
“leave”	:	as a connected peer leaves the room
“reconnect”	:	as a previous disconnected peer reconnects to the current room

Games

“progress”	:	partial updates as connections to other users will be established
”start”	:	as a games starts and the bootstrapping functions gets executed
”pause”	:	as the game loop stops
“resume”	:	as the game loop continues after a “pause”
“end”	:	as the game ends

In addition developers can configure some basic settings like the minimal and maximum amount of players for a game.

5.2.2 Users

Players are special peers.

Before an user can even join a room, its necessary to create a new player. The developers have to declare an action, which is using the ".login" function for the registration. With the creation of a new instance – the framework will automatically check for an alternative default route and attempt to connect to other users in the room.

In addition to the local player, a developer can also access the currently connected peers - which are representations of players on other systems. These copies are merely readable and will be updated automatically by their remote origin. (see 5.2.5 Reactive)

Players and peers are based on same interface and contain following information:

- account : meta information about the user, e.g. the profile name or avatar image
- id : a secured random global unique identifier to distinguish the users
- data : public information and elements for usage in a game
- latency : the average time for sending a message to a peer

Internally keeps each player also a reference to a time stamp according to the moment of their creation. This will be used to define a hierarchy among the users. (see 5.2.3 PeerChain)

To navigate between different rooms players can ".join" a specific address according to a channel or a game. The default route which will be followed as a new player gets created is "lobby/", but can also be changed setting a single address to ".routes".

Inside a room, players can then communicate with other visitors. For this purpose, its possible to either send simple text based messages or media streams. By default the stream are defined solely as audio input, while video record can be enabled as well.
For each type can the developer decide which audience should addressed.
The message or stream can be shared with all connected peers, multicasted to a selected group or just a single reference. On the side of a targeted peer can hooks be set up to handle the retrieved information.

5.2.3 PeerChain

Players create a hierarchy to keep the order and invoke themselves to start a game.

The internal time stamp of each player will be used to define a ranking and keep a consistent order. Starting with a specific structure in an uncertain environment like a P2P network can be useful to ensure a symmetric flow without additional communication and continuous message exchange just for staying synchronized with the same computations.

After a client connected to the server and retrieved a list with available peers, a connection with one of the entries will be created. The credentials will be exchanged and set on the remote system. Then it setup DataChannels and defines services on top. (see 5.2.5 Services)

Over the established connection will then another list received, which contains the peers which are already connected to the new partner. It will be matched with the existing entries. As one isn't already connected with the player, it will create a new connection with this peer and use the established connection as the new communication channel instead of the server.

If the player has created a connection to all peers he is aware of, he will define his position in a in an ordered list. Depending on the initial creation of their player instance, the peers got a different time stamp which will now be compared. Since they are ranked by the lowest value, the player logged in first and stayed at longest will get highest priority.

With re-defining the positions, the “pg.data” will also be restructured. As accessing the data of each user is quite common during the game rendering, the framework provides a shortcut.
This list is contains a reference to the data of each peer and your local player.

Afterwards the players enters the current room and the according event will be emitted.
As a peer leaves the room or disconnects, the “leave” event will be trigger and the positions in the peerchain will be re-define by all players to keep the order synchronized between them.

5.2.4 Initialization

Separate rendering and invoke in order.

For actually starting a game, an initialization function needs to be passed to “game.start()”. The function should be called whenever a player enters the game room. It will check internally if the minimum amount of players got reached by the current number of visitors.

As this is the case, the provided bootstrapping function will be called by the peer at the first position on the peerchain. The passed setup function contains the game code and should use the references defined by the framework for handling data. Existing update and rendering functions should be extracted and executed inside the callback of “pg.loop()”. This callback won't be called immediately, but delayed till the game is setup by every peer.

So the process of running a game happens in two phases: during the setup will the initial game data be defined, while the second finally starts the rendering on all systems.

After the first peer called the bootstrapping function, it will forward the permission for setting up the game on the next peer. The invocation will continue until it reaches the end of the chain. In case that the room is configured to allow users to join the game while its already running, the new player will connect to all peers and request the permission afterwards.
Should previous peers still keep forwarding – will he just be added to the end.

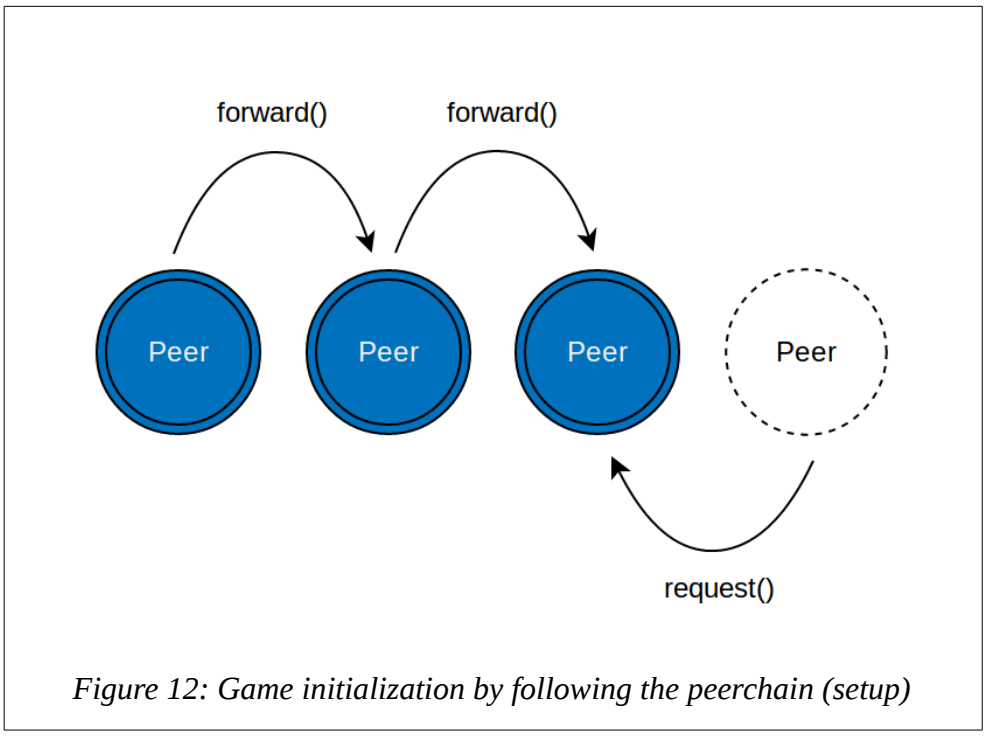


Figure 12: Game initialization by following the peerchain (setup)

The last player of the chain will trigger the start of the rendering loop on the local and remote systems.

5.2.4 Services

Common tasks are defined as defaultHandlers on top of DataChannels.

The framework uses DataChannels to exchange data between two players. As a channel will be created on top of an established PeerConnection, the setup needs to be declared on both sides. As new channels are modifying the existing connection, a re-negotiation will be triggered for each channel.

I defined a set of common tasks, where each will be handled by a different DataChannel.
The default handlers got different labels according to their function:

- init: exchanging basic information on creating the connection
- register: remote delegation to register another peer (proxy packages)
- ping: check the latency and by are responding with a pong
- start: invoke the game on request
- update: sets key and values of a peers remote model
- sync: delegates synchronized changes of the shared object
- message: call and invokes remote text message for the player
- media: handles credential exchange for media streams (audio/video)

All of these are used internally for handling the communication between the peers.

To prevent a delay by triggering multiple re-negotiations, which will require always an additional exchange of credentials, the DataChannels are batched and will be invoked just once. For an easier usage, I defined a “Handler” which is a wrapper on top of a DataChannel.

Their current implementations are limited in their functionalities. According to the specification it will be possible to send binary data over the channels as well, but currently their limited to text based messages. You can still serialize objects into a string or encode arbitrary data as Base64 to transfer them. Unfortunately the channels aren't reliable yet, so in their current implementation they can just buffer around 1kb before they will start dropping data[34]. To work around this limitation, messages which are larger will be automatically chunked into smaller packages. The wrapper itself consists of a stream container, which will assemble these parts on the other side and emit them as one message.

Each handler can therefore trigger hooks on creation, as information will be received (either for each part or as it got assembled as one) and as the underlying DataChannel is closed.

34 https://groups.google.com/d/msg/discuss-webrtc/U927CZaCdKU/O49Q-V2dkSkJ

5.2.5 Reactive

Changing a property on the players data attribute will update remote representations as well.

As mainly game data and states will be exchanged in a multiplayer game, the framework defines “reactive” objects which automatically synchronize their properties between all participants. Each player instance has a “data” attribute, which uses getters and setters to transfer a local change to connected systems. It updates the remote representations, accessible by “pg.peers[id].data” and “pg.data”, through broadcasting the new values.

For the technical implementation it uses the “Object.defineProperty()”[35], to attach functions which will be trigger as a new property and value will be set. Since a player could also add complex objects or arrays with an unknown length, the type of the value will first be determine and define a new “reactive” container as it matches.

To unset a property, its recommended to use “0”, “null” or “false” - as deleting the property directly will also remove the attached handler and no changes will be send to the peers.

35 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty

5.2.6 Backup

Players can reconnect to the last game using a local copy of their former instance.

If a player lost the connection to a game, there is still the possibility to reconnect to it. Each change on the players data will create a snapshot of the current instance, containing the ID, the timestamp and the actual game data. This information will be serialized as a string and stored in the localStorage.

On revisiting the site, the framework will check if an entry is available in the storage and if its backup duration isn't already expired. For the duration will the difference between the current time and the initial time stamp be used. As a valid backup exists, it will extend and overwrite the new player instance which will be created on the login.

The restore option is disabled by default, as not all game would benefit from reconnecting previous players. Enabling the backup can be done through the configurations, where the duration of the storage can also be set.

5.2.7 Synchronized

A shared object will be automatically synchronized between all players.

Multiplayer games consists of data which neither belongs to the local player nor to a remote peer. These data (see 4.2.2 Developers) are often critical, as more than one system can attempt to change persistent values at the same time. Since each system got the permission to access the information and edit them, a conflict between peers with different entries will happen sooner or later.

To solve this problem, the framework got an additional “reactive” object (see 5.2.5 Reactive) which will be shared among all players. The object itself will be synchronized and define a solution for the conflict automatically.

As a player adds locally a new property or changes an existing value on “pg.sync”,
it wont be set immediately. The running game could make decisions based on this result –
even though other peers could have different entries for the same key. Instead of using the value directly, it will therefore be stored into a cache. Since some computations rely on the outcome, by default will the game loop paused until a resynchronized value will be defined.

The next step is sending the value to all connected peers and request their information.

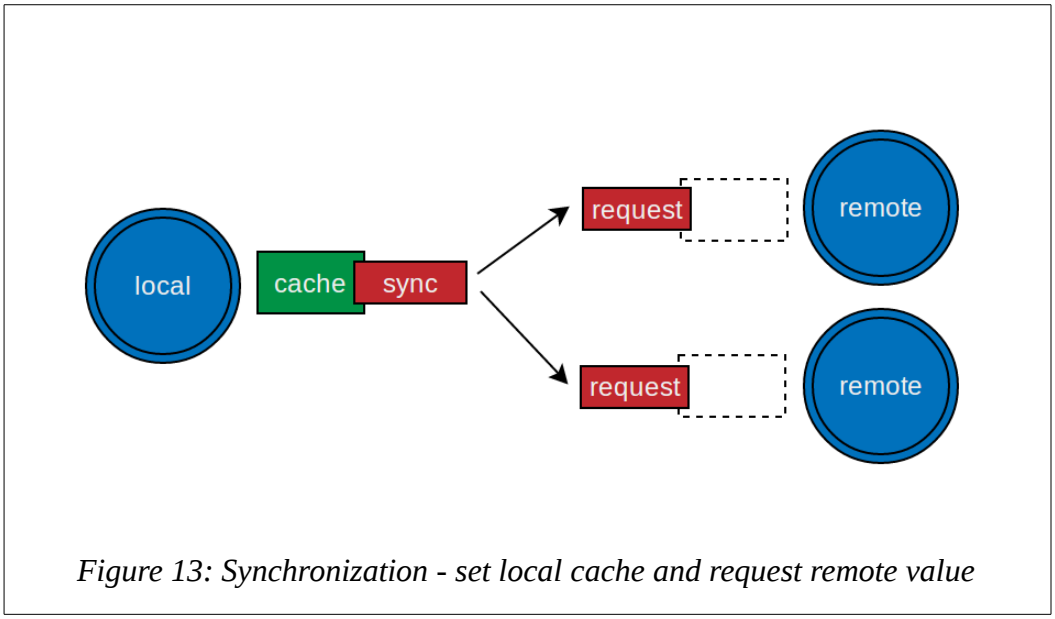


Figure 13: Synchronization - set local cache and request remote value

If they already got an entry into their own cache, it will return this remote value. Otherwise the local value will be set into their cache and used for the respond. The game loop on these remote systems will be paused as well.

The original player keeps track of all peers it sent a request. As he received a message from all of them, the value which is going to be used by every peer will be determined.

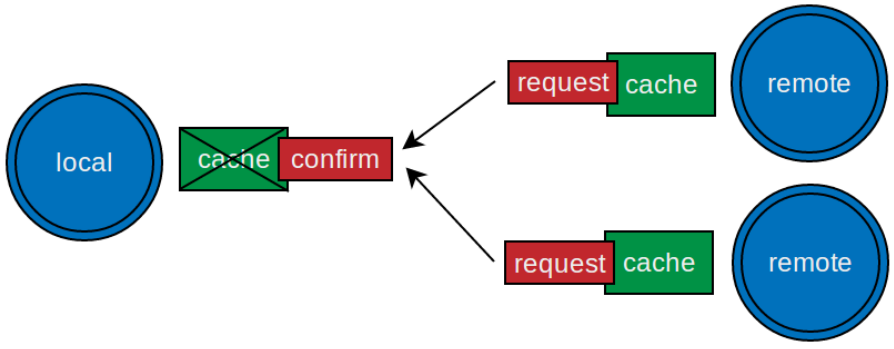


Figure 14: Synchronization - return remote value for confirmation

Different methods can be used to define a proper value. In an earlier version got the value of the peer on the first position of the peerchain picked. Aside its simplicity, this can lead into deceptive results. A better approach is therefore the selection of the value which occurs at most. The frequency of each result will be compared and if there is a tie, just then is the position on the peerchain for these values used as the criteria of priority.

Afterwards the local cache will be invalidated, the resynchronized value will be set and the game loop continues. The same happens on the remote systems.

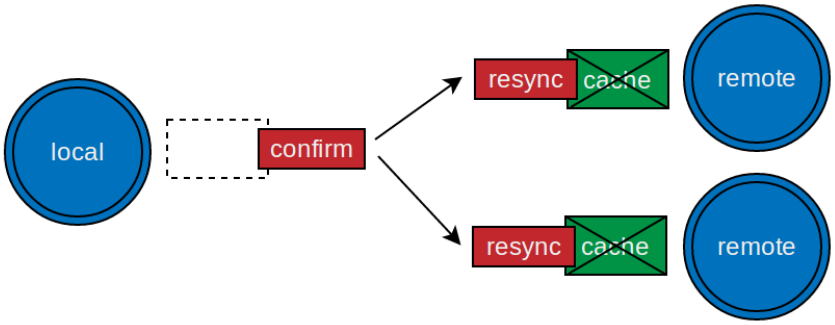


Figure 15: Synchronization - set value and invalidate caches

As the synchronization will be handled between all connected peers, this aspect can be slower than a handling via server. The more players exist which are get their own results, the more conflicts can occur and needs to be handled.

5.3 Challenges

The WebRTC specification is currently still work in progress and the implementations in the browsers can change weekly. Although its exciting to work with new technologies, the disadvantage is the sparse documentation. Outdated information are spread over the internet and can lead to wrong assumptions. For technical problems I can just recommend to use the Google Code Board[36] and the W3C Mailing List[37].

5.3.1 Debugging

Network code with asynchronous execution can be difficult to test. Running two browser side by side on one system will at least simplify the setup, although the latency can't be taken for granted.

While the biggest problem in development are often bugs which can occur but not reproduced, this is even critical in handling networks and P2P communication. For comparing the efficiency of solving a merge conflict for instance, a collision needs to be happening on multiple systems at the same time.

Luckily using the same browser with multiple windows provides another advantage: some resource are shared between the instances and can be used to invoke function calls on both ends simultaneous. LocalStorage and Shared WebWorker provides an opportunity to share messages directly.

Moreover Chrome has with *webrtc-internals*[38] an useful tool to take a look at the internal state of PeerConnections and the transferred data, which helps in tracing bugs with SDP packages.

36 https://groups.google.com/forum/?fromgroups#!forum/discuss-webrtc
37 http://lists.w3.org/Archives/Public/public-webrtc/
38 chrome://webrtc-internals/

5.3.2 Race Conditions

Although the basic concepts of the ICE framework and the creation of PeerConnections are straight forward, the underlying technical requirements aren't obvious until an error will point in this direction. The problem is the order of sending, receiving and using messages.

With creating an “offer” and setting it as the LocalDescription, ICE candidates will be gathered on one system. The SDP package and the candidates have to be transferred to the remote partner, where they can be set and the “answer” can be defined. Sometimes it happened that the following warning occurred and prevented the further construction:

“SetRemoteDescription failed: Called with type in wrong state, type: answer state: STATE_INPROGRESS”

The occasion happened because of the dependency on ICE candidates. These have to be added to the connection before an answer can be created. Even though the candidates were send earlier, it took more time for them to arrive at the other system. To solve this problem will an additional message, which contains information about the expected amount of candidates, transmitted. Just after all candidates arrived and got added, the SDP package will be set and the answer can be created.

5.3.3 Interoperability

The different implementations of the WebRTC specification in the browsers makes it difficult to support all vendors. Even the compatibility of one browser isn't guaranteed: as Mozilla adjusted their implementation towards their official standard, they broke the previous API for DataChannels.[39]

Aside of prefixing the API calls, some core concepts like the “re-negotiation” on the state change of a PeerConnection aren't realized properly. Firefox even throws this warning:

"Renegotiation of session description is not currently supported. See Bug 840728 for status."

Unfortunately it's not possible to rely on error messages either – as they won't be thrown in all cases. Chrome fore instance doesn't support reliable DataChannel based on SCTP yet, while every Channel created in Firefox is using the protocol. As the “offer” will be exchanged and set as the remote description, nothing happens. Neither ICE candidates will be gathered nor warnings will be displayed.

39 http://mozilla.github.io/webrtc-landing/DataChannel_changes.html

6 Conclusion

In the end I will consider the opportunities which will be provided by the framework and the technologies it uses.

6.1 Evaluation

The latest version of the framework is 0.5 and got released on the 23rd July 2013.

Although its not complete yet, it already provides the functionalities to create real-time multiplayer games in Firefox and Chrome. Using PeerConnections and DataChannels to transfer messages directly between two browsers allows developers to create new patterns or try different approaches like the “reactive” data objects.

Using multiple media streams at the same time and running the basic game demo in parallel still left a good impression. The capabilities of using UDP for direct communication will be useful for all kinds of applications and services, especially for ports with a native code base.[40]

As there is currently no real project using the framework, I didn't got the chance to measure real data and verify the advantage of using a direct exchange via PeerConnections. Therefore an upcoming demo will offer more insight about the actual latency, comparing it side by side with a traditional client-server architecture.

40 https://hacks.mozilla.org/2013/03/webrtc-data-channels-for-great-multiplayer/

6.2 Reflection

Looking back at the original requirements on the framework, we can see that it reaches its goals and full fills these aspects:

1.) No additional component has to be required.

The framework consists of one single file without any external JavaScript dependencies. Neither Browser plugins nor specific technologies are mandatory to use it.

2.) *A server shouldn't be necessary.*

Although a public server is provided which deals with the brokering, the developer can also choose to handle the credential exchange by himself and pick any kind of transport channel.

3.) *The majority of desktop users has to be supported.*

As the framework takes care of the different implementations in Firefox and Chrome, both browser are supported. Since they offer automatically updates and cover more than 58% of the current browser market[41], the majority of the desktop user can be reached.

4.) *It should be usable with any kind of rendering.*

As the framework just takes care about the communication and exchange of data, it doesn't affect the rendering at all. Therefore any kind of 2D and 3D visualization can be used.

5.) *Plain data can be exchanged without defining remote procedure calls.*

The “reactive” data property of a player will keep the attributes and values synchronized with their remote representations. A manual setup of functions is not necessary.

41 http://gs.statcounter.com/#browser-ww-monthly-201206-201306-bar

6.3 Future

Upcoming versions of the framework will offer enhanced stability and additional features.

For example will it be possible to define custom routes, which allows developers to uses pattern matching for parsing the query string from a URL and inject parameters to a room. Furthermore are custom handlers going to be supported. This permits the declaration of RPC functions, which can be triggered directly from a remote system.

The development environment will be improved as well. Based on the idea of sharing resources between multiple windows on the same computer, will an offline mode be available which doesn't need a brokering service for exchanging the initial data.

Furthermore will the PeerGaming server be extended to provide together with the framework an interface, which allows other systems to access the current network data. An portation of the back-end in different languages like python and ruby is also planned.

Just recently got WebRTC enabled by default on the Chrome 29 Beta for Android[42], while Mozillas engineers are still working on their implementation for Firefox Mobile – targeting Aurora 24 and the upcoming stable[43]. With this rise of mobile devices supporting PeerConnections and DataChannels, it can be expected that up to 875 million[44] systems at the end of the year will be able to use PeerGaming.

42 http://blog.chromium.org/2013/07/chrome-29-beta-web-audio-and-webrtc-in.html
43 https://hacks.mozilla.org/2013/06/webrtc-comes-to-firefox/comment-page-1/#comment-2151261
44 http://webrtcstats.com/webrtc-forecasts-upgraded-mobile-support-accelerating/

7 Appendix

List of attachments:

- Setup Guide
- File Structure
- Figures

Content on CD:

- the PeerGaming framework (v0.5)	||	/code/peergaming
- the PeerGaming-Server	||	/code/peergaming-server
- a demo which uses the framework	||	/code/example
- a digital version of this thesis	||	/code/thesis
- local backups of material and online resources	||	/material + /links
- installer and binary packages of NodeJS	||	/nodejs
- installer for Googles Chrome	||	/chrome

Setup Guide

As the framework is already built – you can find the script file in the according directory for the distribution (peergaming/dist). To see the script in action and use it later by yourself, a local webserver needs to be setup first. The simplest way is to install the NodeJS framework on your system and run the demo. You can either use the installers from the CD or build it from the source (https://github.com/joyent/node/wiki/Installation). Afterwards you can navigate to the example and run “node index.js”. An instance of the PeerGaming-Server will be automatically started and the static files will be hosted, so you can visit “localhost:2020” with your latest Firefox or Chrome browser.

The latest version can always be found at peergaming.net & github.com/PeerGaming.

File Structure

(each directory has a template which contains insertion instructions for the files in their group)

- **src/**
  - main.js - *basic wrapper for UMD (CommonJS + AMD) and public interface*
  - **core/**
    - _adapter.js - *adjust APIs to normalize different browser behavior*
    - _base.js - *provide the namespace and reserve the old value*
    - _config.js - *default settings and reference for changes*
    - _const.js - *define internal variables and shortcuts*
    - _detect.js - *check browser and environment support*
  - **game/**
    - _game.js - *room handler for a “Game”*
    - _loop.js - *trigger the rendering + update & pause*
    - _peer.js - *container for storing data and account*
    - _player.js - *instance of the player*
    - _sync.js - *shared object between players*

meta/

- _auth.js - *identification for external services*
- _backup.js - *restoring previous informations*
- *_channel.js - room handler for a “Channel”*
- _info.js - *general information to access*
- _login.js - *register the player*
- _media.js - *using media input to share with others*
- _router.js - *parsing the URL and defining the path*
- _watch.js - *notification by internal messages*

network/

- _data.js - *datachannel based on a peerconnection*
- _handler.js - *wrapping datachannel and chunks*
- _manager.js - *communication between players*
- _service.js - *set default handler for remote execution*
- _socket.js - *client-server transport for initial exchange*

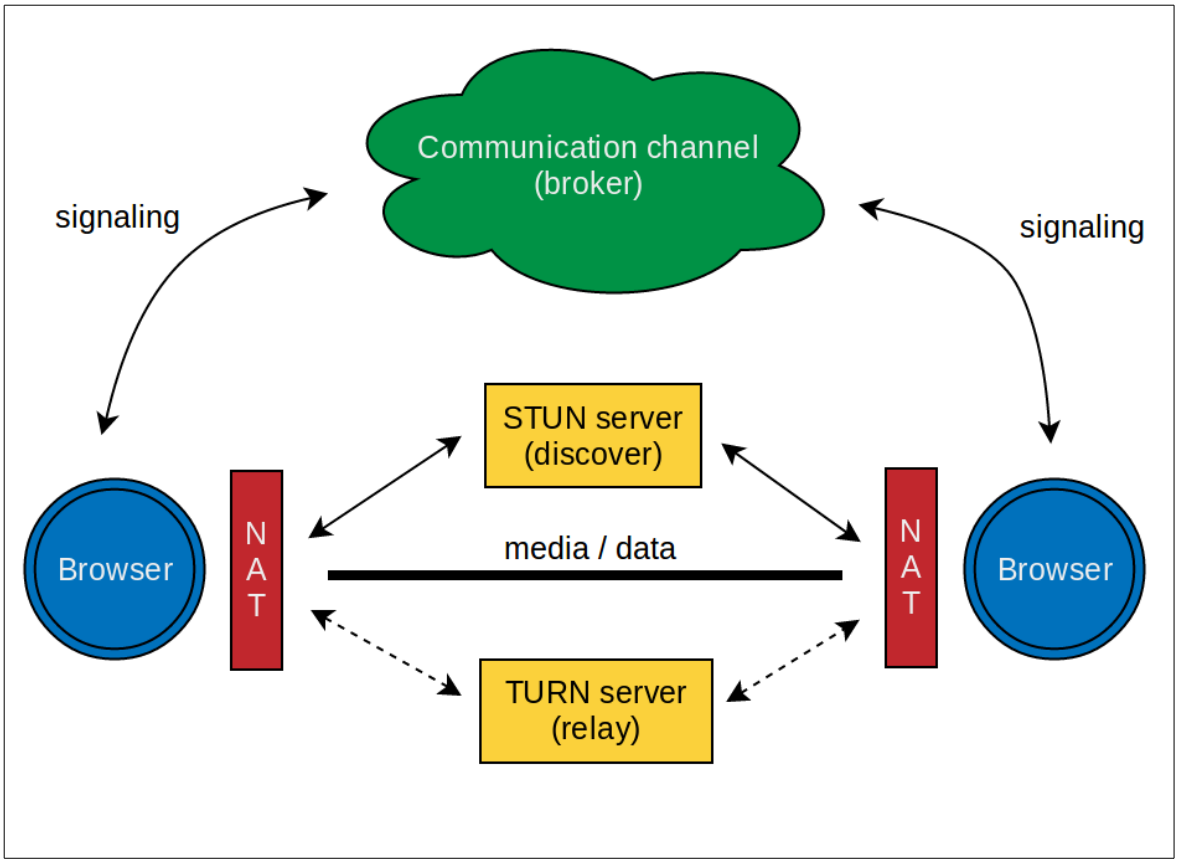
struct/

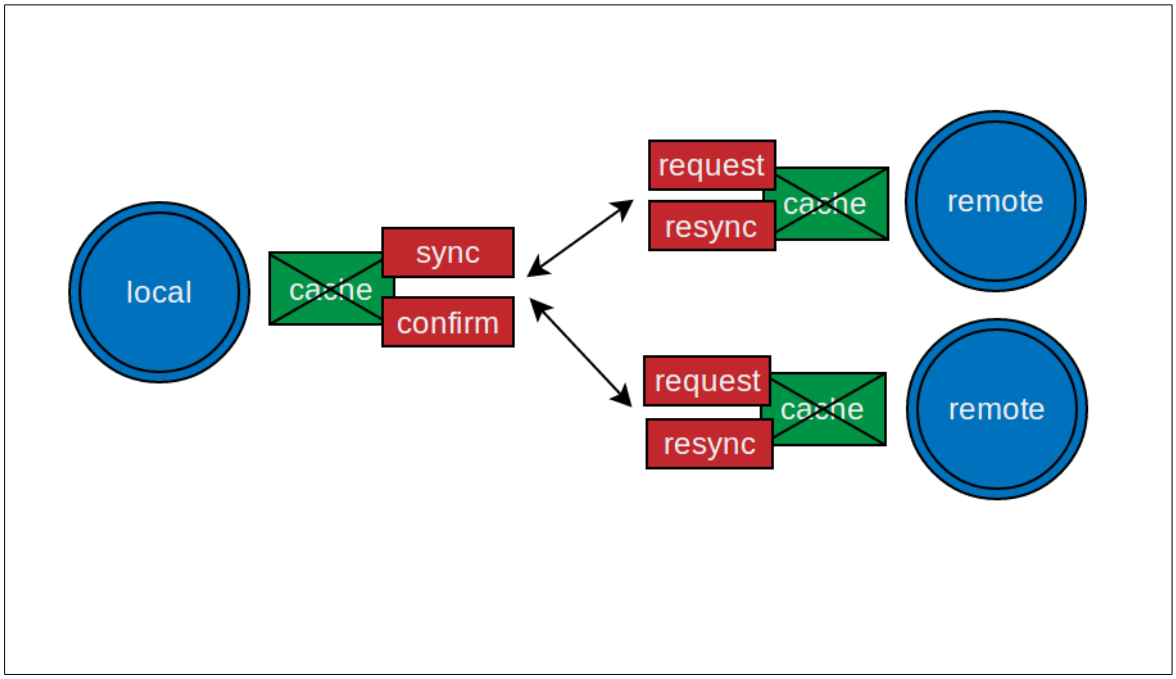
- _connection.js - *define peerconnections and credentials*
- _emitter.js - *implement the publish-subscribe pattern*
- *_reactor.js* - *a reactive object which uses getter/setter*
- _stream.js - *transfer data and handling chunks*

utils/

- _common.js - *general standard helpers*
- _debug.js - *helper to trace and fix bugs*
- *_local.js* - *sharing local resources*
- *_specific.js* - *custom helpers for the framework*

Figures


Communication channel
(broker)
signaling
signaling
STUN server
(discover)
Browser
NAT
media / data
NAT
Browser
TURN server
(relay)


request
resync
cache
remote
local
cache
sync
confirm
request
resync
cache
remote

8 Glossary

AJAX – Asynchronous JavaScript and XML

- a technique to send HTTP requests from a browser without refreshing the site

AMD – asynchronous module definition

- a format to define module dependecies for loading JavaScript files in the browser

API – application programming interface

- an abstraction to access internal processes of a specific software

CSS – Cascading Style Sheets

- a declarative language to visualize content via colors or shapes e.g. with HTML

CU-RTC-Web – Customizable Ubiquitous Real-Time-Communication over the Web

- Microsofts proposed specification instead of WebRTC (doesn't use SDP)

Closure

- a containment of the state even after receiving a return value of a function

CommonJS

- module definition which is used in NodeJS and libraries like Browserify

DOM – Document Object Model

- an abstraction on top of rendered HTML elements which provides a graph structure

DTLS – Datagram Transport Layer Security

- secures packaged orientated transports like UDP (based on TLS)

ECMAScript

- specification for a scripting language (on which JavaScript is based)

HTML – *Hypertext Markup Language*

- a language to define semantic structures and information, rendered by a browser

ICE – *Interactive Connectivity Establishment*

- exchanging candidates & SDP packages via an “offer-answer” scheme

IETF – *Internet Engineering Task Force*

- an organization which defines and promotes internet standards

JSEP – *JavaScript Session Establishment Protocol*

- a defined interface to control the ICE framework via a JavaScript API in the browser

JSON – *JavaScript Object Notation*

- a file format in form of a serialized JavaScript object

NAT – *Network Address Translation*

- mapping a public IP address to a group of computers

NodeJS

- a framework which uses the V8 JavaScript engine to run code

P2P – *peer-to-peer*

- a network consisting of out of peers without a communication server

PaaS – *Platform as a Service,*

- a business model that offers a cloud based solution for an IT problem

RIA – *Rich Internet Application*

- a web application with desktop characteristic functions and behavior

RPC – *remote procedure call*

- a pattern which allows to call functions from a remote system invoked by messages

RTP – *Real-Time Transport Protocol,*

- a format for streaming data over IP networks

SCTP – *Stream Control Transmission Protocol,*

- a transport layer protocol based on of UDP, but with congestions control like TCP

SDK – *software development kit*

- a set of development tools for a specific environment

SDP – *Session Description Protocol*

- a format which is used to describe information about media streams

SRTP – *Secure Real-Time Transport Protocol*

- a profile of RTP which can authenticate messages

SSE – *Server-Sent Events*

- a technology to keep a HTTP connection open and receives messages by a server

STUN – *Session Traversal Utilities for NAT*

- a protocol which allows to retrieve the public IP address inside a private network

TCP – *Transmission Control Protocol*

- a stream based transport layer which handles reliability and structured data

TLS – *Transport Layer Security*

- a cryptographic protocol which secure communication over the internet

TURN – *Traversal Using Relays around NAT*

- an extension of STUN which can in addition be used to relay data

UDP – *User Datagram Protocol*

- a transport layer which sends packages over an IP network (unreliable)

URL – *uniform resource locator*

- address often used to identify and reach a website

W3C – *World Wide Web Consortium*

- an organization which defines standards for the web

WebRTC – *Web Real-Time Communication*

- a set of APIs which deals with real-time communication in the browser

XML – *Extensible Markup Language*

- a language to describe an respresent data structures

9 Sources

List of source:

- Literature
- Protocols
- Material
- Websites
- Projects

Literature

[01] APIs and RTCWEB Protocols of the HTML5 Real-Time Web, Second Edition
ISBN: 978-0-9859-7883-9
http://webrtcbook.com/

[02] “Learning JavaScript Design Patterns”
ISBN: 978-1-4493-3181-8
http://shop.oreilly.com/product/0636920025832.do

Protocols

[03] RTC Web - http://tools.ietf.org/html/draft-ietf-rtcweb-overview-06
[04] DataChannel - http://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-05
[05] JSEP - http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03
[06] ICE - http://tools.ietf.org/html/rfc5245
[07] SDP - http://tools.ietf.org/html/rfc4566

Material

[08] – Presentation: “Aktuelle Themen IMI: Digital Game Based Learning – 01”

[09] – Presentation: “Verteilte Systeme – 1: Einführung”

[10] – Presentation: “Verteilte Systeme – 5: Verteilte Daten”

[11] – Chart: “Essential Facts about the Computer and Video Game Industry – 2013”

[12] – Whitepaper: “Turbulenz Games Platform: Technology Introduction”

[13] – Presentation: “22nd Chaos Communication Congress – peer to peer under the hood” (https://www.youtube.com/watch?v=LXAW4HwFt58)

Websites

[14] – http://www.webrtc.org/

[15] – http://www.html5rocks.com/en/tutorials/webrtc/basics/

[16] – https://developer.mozilla.org/en-US/docs/WebRTC

[17] – http://docs.webplatform.org/wiki/apis/webrtc

[18] – http://bloggeek.me/resources/webrtc-series/

[19] – http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/

[20] – http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking/

Projects

[21] – http://peerjs.com/

[22] – https://github.com/peer5/sharefest

[23] – https://www.webrtc-experiment.com/

Erklärung zur Bachelor-Thesis (Affidavit)

Hiermit versichere ich die vorliegenede Bachelor-Thesis ohne Hilfe Dritter, nur mit den angegeben Quellen und Hilfsmitteln angefertigt zu haben. Diese Arbeit wurde zudem bislang weder in gleicher, noch in ähnlicher Form einer Prüfungsbehörde vorgelegt.

Berlin, den 22.07.2013

_____

(Stefan Dühring)