

MINISTRY OF SCIENCE AND EDUCATION OF THE REPUBLIC OF
KAZAKHSTAN

Non-Profit Joint Stock Company
ALMATY UNIVERSITY OF POWER ENGINEERING AND
TELECOMMUNICATIONS

Department Telecommunication systems and networks

«Admitted»

Head of the Department Baykenov A.S.

d.t.s., professor

(Surname and initials, degree, rank)

« » 20 y.
(sign)

DIPLOMA PROJECT

Theme: Designing a video and audio broadcasting
systems for mobile devices

Specialty: 5B071900 – Radio engineering electronics and telecommunications

Implemented by: Khalil Y.B. ICTe 14-9
(Student's surname and initials) group

Scientific Supervisor: Senenyakin N.V. senior lecturer, PhD
Cleef (Surname and initials, degree, rank)
«30» 05 2018 y.
(sign)

Advisors:
of Economy section:
associate Professor, PhD, Tuzelbayev B.I.
AF (Surname and initials, degree, rank)
«25» 05 2018 y.
(sign)

of Life activity safety section:
senior lecturer, PhD Begimbayeva A.S.
AF (Surname and initials, degree, rank)
«25» 05 2018 y.
(sign)

of Computer Science section:
Senenyakin N.V., senior lecturer, PhD
Cleef (Surname and initials, degree, rank)
«30» 05 2018 y.
(sign)

Standards compliance controller: Chezhimbayeva K.S., docent
AF (Surname and initials, degree, rank)
«30» 05 2018 y.
(sign)

Reviewer: _____
(Surname and initials, degree, rank)
« » 20 y.
(sign)

Almaty 2018 y.

MINISTRY OF SCIENCE AND EDUCATION OF THE REPUBLIC OF
KAZAKHSTAN

Non-Profit Joint Stock Company
ALMATY UNIVERSITY OF POWER ENGINEERING AND
TELECOMMUNICATIONS

Institute of Space Engineering and Telecommunications (ISET)
Specialty: 5B071900 – Radio engineering electronics and telecommunications
Department: Telecommunication systems and networks

ASSIGNMENT

For diploma project implementation

Student: Khalil Yermek Bakhytaly
(name, patronymic and surname)

Theme: Designing a video and audio broadcasting
systems for mobile devices

Approved by Rector order № 155 of « 23 » october 20 17 y.

Deadline of completed project: « 25 » may 20 18 y.

Initial data for project, required parameters of designing result, object initial data:

You have to install browser on your personal computer.
the concept of designing of video and audio broadcasting
considers learning WebRTC technology. And as a basis
of programming language "javascript" should be studied.

List of questions for development in diploma project or brief content:

The creation of audio and video broadcasting
systems for mobile devices are considered in this
diploma project. The calculations of the server load
are carried out. the life activity safety includes the
calculations of natural lightning and air exchange. the
technical and economical basis of the project is given.

List of illustrations (with exact specifying of mandatory drawing):

Browser-to-browser technology; Technologies used in WebRTC;
 Developed architecture for data exchange in the application
 of video conferencing; the algorithm of preparation of the
 before generating the signal data; the algorithm for
 allocating sockets and processing of the buffers on the
 server; Algorithm for working with the data buffer;
 MediaStream carries one or more synchronized tracks;
 List of constraints; RTCPeer Connection API; Cost structure
 of software development

Recommended main references:

Chan, T. et al. Studying with the cloud: the use of online
 web-based resources to augment a traditional study group
 format / T. Chan, S. Sennik, A. Zake, B. Trotter // CJEM - 2014
 Мечуев С. Модульная разработка программного обеспечения / С. Мечуев //
 Модульное обучение. - 2007. - №12 - с. 48-53
 Civanlar, M.R. et. al. Peer-to-peer multipoint video conferencing on
 the Internet // Signal Processing: Image Communication - 2005 - T20

Project adviser with corresponding sections specifying:

Section	Advisor	Dates	Sign
Life activity safety	Beginbetova A.S.	2.03.18-25.05.18	AB
Economy	Tuzelbayev B.I.	5.04.18-25.05.18	BT
Main section	Semenyakin N.V.	10.02.18-25.05.18	NS
Computer science	Semenyakin N.V.	10.02.18-25.05.18	NS


SCHEDULE of diploma project implementation

[illegible]

Assignment issue date « 10 » January 2018 y.

Head of Department: _____ Baykenov A.S
(sign) (Surname and initials)

Scientific Supervisor: _____
(sign) _____
Semenyakin N.V.
(Surname and initials)

Assignment submitted for implementation:  Khalil Y.B.
(sign) (Surname and initials)

Аңдатпа

Дипломдық жобада мобильді құрылғалыр үшін мультимедиа контентінің(презентация, аудио және видеохабар) жеткізуін жүзеге асыратын ақпараттық жүйені жобалау қаралған. Жобада жүйе өнімділігін бағалау үшін оңтайлы параметрлері қаралған.

Жоба төрт бөлімнен тұрады, олар мультимедиа контентін жеткізу әдісін жүзеге асыратын ақпараттық жүйе құру, техникалық есептеулер, еңбек жағдайларын және экономикалық тиімділігін талдау. Бұл жоба Қазақстанда ғана емес бүкіл әлемде талапты болатыны анықталды. Жобадағы проблемаларды шешудің өзіндік тәсілі мен бірқатар артықшылықтары бар.

Экономикалық бөлімінде табыс жолдары, алдағы даму жолдары бойынша жұмыс атқарылды. Бұл экономикалық орындылығын дәлелдейді.

Аннотация

В дипломном проекте рассмотрено проектирование информационной системы, реализующей доставку мультимедийного контента (презентация, аудио и видеотрансляция и т.д.) для мобильных устройств. В проекте рассмотрены параметры для оценки производительности системы.

Проект состоит из четырех глав, которые описывают основные методы выбора решения по созданию информационной системы, реализующей доставку мультимедийного контента, технических расчетов, анализа условий труда и экономической выгоды. Определенно, что данный проект будет востребован, как на рынке Казахстана, так и во всем мире. Он имеет оригинальный подход к решению имеющихся проблем и имеет ряд преимуществ.

В экономической части был проработан способы получения дохода от приложения, для дальнейшего развития, что доказывает экономическую целесообразность.

Annotation

The diploma project considered the design of an information system that implements the delivery of multimedia content (presentation, audio and video broadcasting, etc.) for mobile devices. The project studies the parameters for assessing the performance of the system.

The project consists of four chapters, which describe the main methods of choosing a solution for creating an information system that implements the delivery of multimedia content, technical calculations, analysis of working conditions and economic benefits. Definitely, that this project will be in demand, both in the market of Kazakhstan, and in the whole world. The application has an original approach to solving problems and has several advantages.

In the economic part, ways to generate income from the application, for

further development have been worked out, which proves economic feasibility.

Content

Introduction	
1 Analysis of the current state of WebRTC technology	8
1.1 What does RTC term stands for?	8
1.2 Advantages of WebRTC	9
1.3 Methods of data transmission in peer-to-peer web applications	10
1.4 Standards and Development of WebRTC	11
1.4 Usage of WebRTC in present time	12
2 Practical realization of the project	14
2.1 Formulation of the problem	15
2.2 The architecture of the interaction of the modules of the web application	16
video conferencing	
2.3 Algorithms for establishing connections between clients via WebRTC	
protocol	
2.4 Acquiring Audio and Video with getUserMedia	22
2.5 Real-Time Network Transport	25
2.6 RTCPeerConnection API	25
2.7 Establishing a Peer-to-Peer Connection	26
2.8 Multiparty Architectures	27
2.9 Listing of the program	29
3 Life activity safety section	40
3.1 Assessment of the forthcoming physical and mental load	40
3.2 The calculation of the integral scoring after optimization	44
3.3 Conclusion of the section	47
4 Economical part	47
4.1 Calculation of the cost of software development	47
4.2 Calculation of the complexity of software development	48
Conclusion	56
List of abbreviations	57
List of references	58
Appendix A Listinng of programming of CSS part of the application	60
Appendix B List of programming of server part	61
Appendix C List of programming of getUserMedia function	62
Appendix D Anti-plagiarism certificate	
Appendix E Electronic version of the diploma work and demonstration	
erials (CD-R)	
Appendix F Handouts (A4 format – 13 pages)	

Introduction

In the development of video conferencing systems, the main attention is paid to the methods of multimedia data processing, methods of data transmission, optimization of the architecture of the connection of client applications. The implementation of video conferencing systems in mobile devices imposes its limitations on the quality and volume of information processed due to insufficient computing and network embedded resources of mobile heterogeneous devices.

Insufficient bandwidth of communication channels for the transmission of rapidly growing amounts of transmitted multimedia data requires the development of new methods of transmission. In multi-user video conferencing systems, multiple identical multimedia data must be transmitted at the same time between all participants in a communication session, which significantly increases the volume of flows and the load on the server.

In addition to these disadvantages, there is a problem of embedding applications for use in larger systems. This problem is due to the fact that all existing applications are complete software and do not have the functionality for integration into third-party systems.

Therefore, the relevance of the development of architectures, algorithms and software for automatic processing of multimedia data streams in peering (peer-to-peer) web video conferencing applications, providing a reduction in the amount of data transmitted and the ability to build speech and multimodal interfaces for infocommunication applications, is confirmed by the lack of cross-platform software and hardware heterogeneous client applications that support multi-channel communication of remote subscribers.

The aim of the diploma work is development of an information system that implements the delivery of multimedia content (presentation, audio and video broadcasting, etc.) for mobile devices.

For achieving this aim, it is necessary to solve the following tasks:

- Analysis of modern architecture of video conferencing systems, as well as methods and software for multichannel processing of audio-visual and service data streams;
- Development of server architectures in peer-to-peer video conferencing systems that reduce the amount of data transferred and reduce the consumption of server client applications;
- Development of algorithms for server parts of the video conferencing system and establishing connections between clients that provide distribution and processing of their data flows to clients;
- Development of software for server parts of the web application for screen sharing, providing cross-platform and multi-channel communication sessions between heterogeneous devices of distributed subscribers.

1 Analysis of the current state of WebRTC technology

Video conferencing applications that run on desktop computers in most cases have the necessary resources to process data, but with a large number of incoming and outgoing streams, problems may occur due to overload of the Central processor, RAM and graphics card of the device. On mobile devices, the situation is aggravated by the lack of resources necessary for processing large amounts of data and small displays that are not able to correctly display more than two participants at the same time. Specialized collaboration systems, in addition to transmitting audio-visual data streams captured by microphones and video cameras, allow the exchange of additional multimedia information, such as the transmission of the current presentation slide, as well as the possibility of joint editing of documents and handwritten drafts.

These and other problems of designing video conferencing systems are discussed in the first Chapter. When analyzing the architecture, capabilities and number of users of existing video conferencing systems, the most promising were chosen peering systems that provide a reduction in the amount of data transmitted and the ability to build voice and multimodal interfaces.

1.1 What does RTC term stands for?

The topic of real-time communications has attracted considerable attention, but generally speaking, the term RTC is not new – it has long been so called any way of interaction in which you can ignore delays. RTC includes the exchange of data in full-duplex (bidirectional) and half duplex (subscribers use the carrier alternately) modes, as well as data transmission via peer-to-peer networks (P2P). An unaddressed broadcast transmission (Broadcast) and its subset (Multicast) addressed to a limited group of subscribers does not correspond to the representation of the RTC — they do not exchange in both directions. With the advent of the Internet to the traditional means of the category of RTC added instant messaging technology (IM), messaging at the application level in the networks of IRC (Internet Relay Chat), various technologies of teleconferencing, at the same time, e-mail and blogs are not attributed to the RTC due to the noticeable delay.

In the creation of the "new telephony" the main role will be played not by services, as it has been so far, but by applications, which, for example, can find subscribers by their names, freeing subscribers from binding to numbers. It is very good that the restructuring to a new telephony is limited exclusively to applications and does not affect the actual technology of signal transmission — physics and logic of communications can develop autonomously. Ultimately, we are talking about the adoption of new standards, which are comparable in importance to the TCP/IP Protocol family — the future is seen in peer-to-peer networks, well-known to users of decentralized file-sharing networks. Simplistically, their essence is that the server only establishes a connection between the exchange participants, and then the rest of the exchange is given to the subscribers.

Most often WebRTC is presented as a sum of technologies of combining two browsers (Figure 1.1) and their transformation into communication devices, but this

is only partly true. In the first stages, computers or gadgets capable of supporting traditional browsers will really be combined in this way, and the first WebRTC applications will be limited to pair or group communication — for example, such as teleconferences. But, theoretically, nothing prevents to use the same principles of WebRTC in TVs, cars, cameras and other professional and household appliances, where not necessarily the presence of a person, that is WebRTC will be in demand on the Internet of things.



Figure 1.1 – Browser-to-browser technology

1.2 Advantages of WebRTC

Indeed, WebRTC has a number of advantages: "machine browsers" are not tied to a specific OS, do not need to download plugins, and downloadable browsers can save objects of the Internet of things from one of the potential threats: if the object to make non-renewable, it is easy to become the subject of hacker attacks. If everything goes as it seems today, we can assume that we are on the verge of serious changes in communications in General, and as for browsers, the change in their functionality can be compared with what happened in 1993, when the opportunity to reproduce images by means of browsers opened — at this moment the Web ceased to be a purely text space.

The idea appeared on the market a couple of years ago. Although the idea of using peer-to-peer networks for content delivery existed before, they all used additional SOFTWARE, which prevented active development. For example, peer-to-peer video delivery tried to make a social network "Vkontakte". When trying to view the video, users with Flash Player version 10.1 or later were asked to allow connection to a peer-to-peer network (implemented with Flash Player 10.1 and Adobe Cirrus). And the popular social network is not the only resource that tried to reduce the cost of video broadcasting. Another interesting example is the peer-to-peer network, organized by the order of CNN by the Danish company Octoshape using the plug-in for Flash Player. But in both cases, users were wary of the innovation. Many thought that they were trying to install some spyware. And with the development of WebRTC, we talked about a purely browser-based

implementation that works without additional SOFTWARE. PeerCDN is perhaps the first example of using WebRTC to deliver "heavy" content (not just video, but also images, as well as files from storage).

PeerCDN was built using JavaScript. Thanks to the script embedded in the page, fragments of "heavy" content already loaded into the browser buffer could be transferred to other users, thereby reducing the load on the Central server. The system allowed to save significantly on the cost of traffic (especially at peak loads). According to the "laboratory" tests conducted by the startup, the savings could reach 90%. But, apparently, PeerCDN had no commercial results (until the last moment The decision was at the stage of public beta). At the end of 2013, the technology together with the developers came under the leadership of Yahoo! (conditions of transition are not disclosed). At the same time, information about the models of monetization of PeerCDN has not yet been received. Third-party observers suggested that the developers will use their servers as paid trackers and authorization centers when creating a peer-to-peer network. But these assumptions were neither confirmed nor refuted.

Most users do not go into the technical details of the peer-to-peer network. Although the forums devoted to similar solutions, you can find comments dissatisfied with the fact that the organizers of the broadcast without explicit permission plan to use the resources of their workstations and channels (which may well imply payment for megabytes).

The WebRTC technology itself is quite new. So far it is not abused, and it is allowed in browsers by default. Perhaps if this becomes a problem for users, browser developers can introduce some kind of setting, like for JS, that will allow to refuse the use of WebRTC. But so far there is no such practice.

1.3 Methods of data transmission in peer-to-peer web applications

Despite the rapid pace of development of Internet technologies, there are many problems associated with the streaming of video and audio. In many ways, these problems arise due to insufficient bandwidth. Since video systems require large network resources even for video transmission between two participants, support for multi-user video conferencing is extremely difficult.

Now hundreds of thousands of users can simultaneously use peer-to-peer networks. Common practice in p2p video streaming systems is the Association of participants viewing the same content in "swarm" and redistribution of parts of video content exclusively between the members of this swarm. For such a channel-isolated structure P2P systems characteristic of the delay for switching channels and the backlog of content playback related to the churn of the channel and the imbalance of the number of receiving and relaying nodes. In General, global P2P networks with a channel-isolated structure currently have serious performance problems, which will become more serious with the increase in the number of channel users.

The performance of video streaming methods in P2P networks also depends on the configuration of the network itself, its topology, heterogeneity of network resources of subscribers, the bandwidth of their communication channels. Unlike

joint download files where small bandwidth leading to slow downloading, with streaming video of low-speed connectivity becomes a real problem. Video compression, which allows to reduce the channel load without a significant increase in the load on the end user device when encoding/decoding the signal, also remains relevant.

1.4 Standards and Development of WebRTC

Enabling real time communication in the browser is actually an ambitious undertaking, and arguably, one of the most significant additions to the web platform since its very beginning. WebRTC breaks away from the familiar client-to-server communication model, which results in a complete re engineering of the networking layer in the browser, and also brings a whole new media stack, which is actually necessary to enable efficient, real time processing of video and audio.

As a result, the WebRTC architecture consists of over a dozen different standards, covering both the application and browser APIs, as well as a variety of protocols and data formats required to make it work:

- Web Real Time Communications (WEBRTC) W3C Working Group is actually responsible for defining the browser APIs.

- Real Time Communication in Web browsers (RTCWEB) is actually the IETF Working Group responsible for defining the protocols, security, data formats, and all other essential elements to enable peer-to-peer communication in the internet browser.

WebRTC isn't a blank slate standard. While the primary purpose of its is usually to enable real time communication between browsers, it's also designed such that it may be integrated with existing communication systems: voice over IP (VOIP), various SIP clients, and perhaps the public switched telephone network (PSTN), just to name just a few. The WebRTC standards don't define any specific interoperability requirements, or perhaps APIs, but they do attempt to reuse the same concepts and protocols where possible.

Put simply, WebRTC isn't just about bringing real time communication to the browser, but also about bringing all of the capabilities of the Web to the telecommunications world - a 1dolar1 4.7 trillion industry in 2012! Not surprisingly, this's a significant growth and one that many existing startups, businesses, and telecom vendors are following closely. WebRTC is a lot more than just another browser API.

WebRTC uses two audio codecs, the G 711 and OPUS, and the VP8 and H. 264 video codec. They are shown in Figure 1.2.

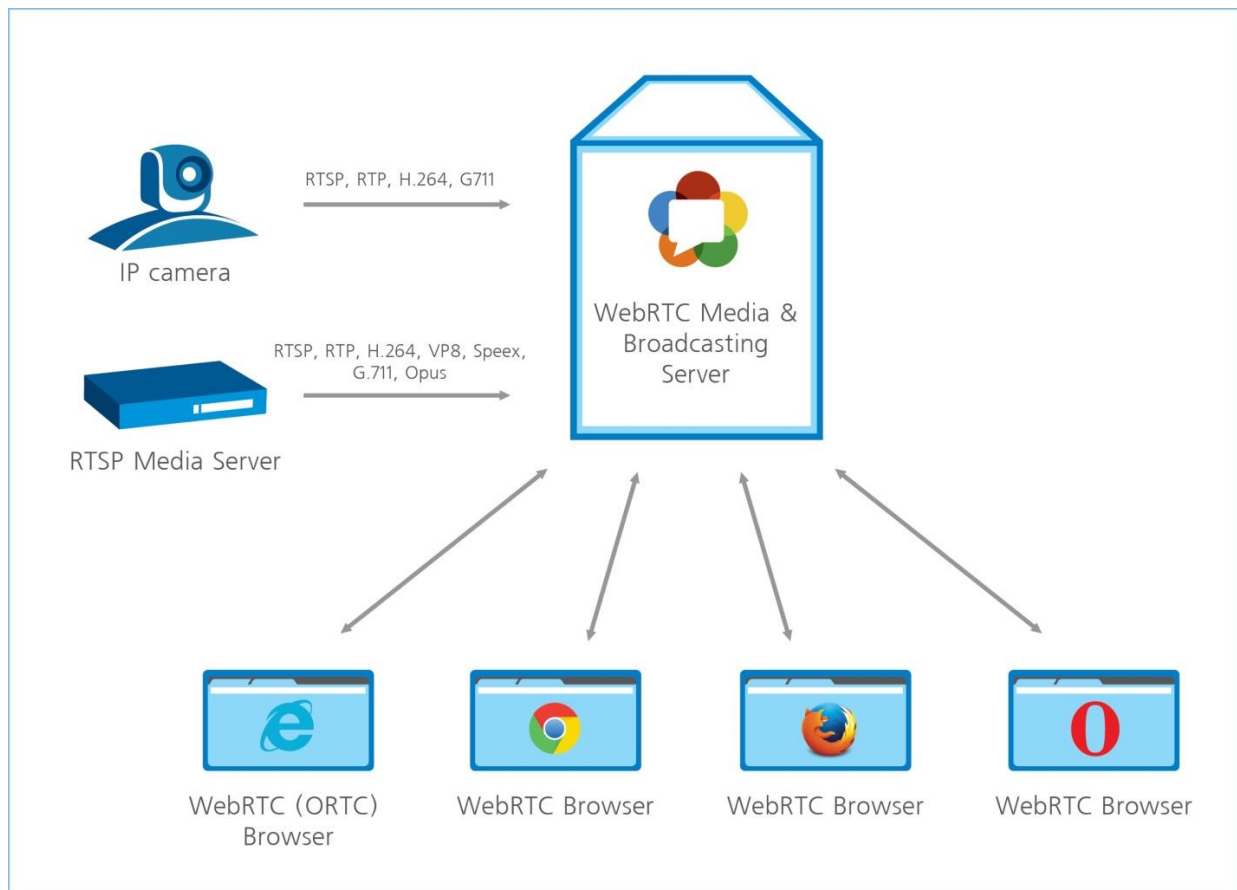


Figure 1.2 – Technologies used in WebRTC

1.5 Usage of WebRTC in present time

Project statistics show that the solution allows you to save up to 70% of traffic at peak times, and at normal times - up to 30%. At the same time, for the normal operation of the peer-to-peer network, simultaneous viewing of video by hundreds of users is enough. And to achieve 50% savings makes it possible for 500 active viewers.

So at the moment, we are aware of four competitors offering similar technologies.

Peer5 is a tool for building a decentralized network for the delivery of bulk content. Unfortunately, there is no detailed description of its principal features on the solution website, except for mentioning the possibility of using not only for video, but also for audio, online games and delivery of three-dimensional images. However, part of the project code is distributed under the Open Source license.

In 2013, the Peer5 solution became part of Kaltura's technology platform. In our country, Kaltura is known for a platform that provides users with the ability to create their own videos based on media content distributed under the Creative Commons license, including through resources such as YouTube. In addition, Kaltura develops solutions for publishing and monetizing media content (in particular, OTT-platform). Integration of Peer5 into the Kaltura platform is interesting because the companies jointly demonstrated the capabilities of The technology, simultaneously connecting users from the USA, Europe, Asia and Australia. The collected statistics showed that about 90% of the traffic went through

the peer-to-peer network. The average time to start a video in the user's browser has been reduced from" average for the industry " 2.2 seconds to 1.5 seconds (this parameter is considered important thanks to a report of analysts from the California company Conviva, according to which the average user refuses to view the video somewhere between 2 and 3 seconds waiting for the launch). Apparently, the result achieved in the course of the demonstration is now 90% and is used by similar projects in their marketing materials.

Startup does not report about other clients or implementations, as well as about monetization schemes.

Swarmify allows you not only to work with video, but also to preload other content, for example, pages of the site, which are likely to be transferred to a mass user (the founders of the startup call this opportunity "predictive download"). The site solutions reports four customers: the online store MakeUseOf, advertising Agency, Digital MGMT, resource SwimSwam and some of the company Entertain DL. In addition, it reveals the principles of monetization: the tariff plan is determined by the total amount of traffic transmitted through the peer-to-peer network for the month. Up to 250 GB per month is free, up to 10 TB - for \$ 99 per month, large volumes - according to the individual tariff plan.

On the solution website, there is a demonstration with statistics, which allows you to clearly see the principle of its operation.

Viblast is a solution that specializes in the transmission of live HD video to mobile devices and web clients. Unlike other projects, developers focus on the ease of integration (which comes down to integrating a pair of libraries into mobile applications and inserting a number of lines of code) in the advertising of this startup. Unlike the competitors described above, Viblast offers demo applications for iOS and Android.

The option is offered as a service. Monetization is based on the amount of saved traffic (i.e. traffic transmitted through the peer-to-peer network). At the moment, the solution supports only "live" broadcasts, but announced support for video on demand, which should appear in the third quarter of this year.

The solution also has a demo. The service clients are not reported.

StreamRoot is another solution that works with both live and on-demand video streams, supporting adaptive streaming. At the moment, the developers have implemented plugins for JWplayer, Flowplayer and VideoJS, but are ready to combine their product with any player based on HTML5.

The developer States that its customers are France televisions, orange operator and L'equipe. Monetization of the service is based on the volume of content transmitted per month, as well as the average number of viewers on the site. A free SaaS distribution model has been announced, but it is still in development mode (interested parties are invited to subscribe to the newsletter for further details).

Interestingly, for all of these solutions offer a rather scant explanation of the details of the technology on the site, preferring to conduct sales through a 30-day trial version. Although all of them have something to tell (in particular, about the encryption of the transmitted content).

These companies are not the only market participants who have paid attention to peering. Not everyone bets on WebRTC, but the idea itself, as they say, "is in the air."

Netflix is engaged in development in this area, Akamai also launched some test projects. As I said earlier, V Kontakte had an attempt to use peer-to-peer video transmission from Adobe. However, then they refused this idea (perhaps, at their volumes it is not very effective, since the number of videos there is huge and the consumption is smeared on different rollers; in addition, Adobe had to be additionally allowed peering transmission, which, I think, frightened off many). We plan to present our solution at IBC (stand 14.D01 in the connected world pavilion) in September this year. As far as I know, our colleagues from Viblast will be at the exhibition. So we look to the future with optimism.

Based on the proposed principles of optimization of multimedia data exchange methods in the screen sharing application, a peer-to-peer architecture of direct audio and video transmission between the client parts was developed, presented in the next section. The process of forming client web pages and establishing communication with the server via WebSocket Protocol is described. The next section discusses the developed algorithms for establishing connections between clients using the WebRTC Protocol. Modern approaches to the development of communication protocols between different multichannel devices and an overview of advanced network technologies are presented.

2 Practical realization of the project

2.1 Formulation of the problem

The problem of audio and video data transmission in peer - to-peer web video conferencing applications is considered. When multiple client and server parts of a video conferencing application communicate with each other, the WebRTC Protocol can cause partial or complete loss of signal data that prevents clients from connecting. The proposed architecture of transmission and storage of "signal" data on the server provides buffering and subsequent processing of "signal" data, eliminating their loss and maintaining interaction between groups of clients.

For the formal description of the problem of synthesis of architecture of peer-to-peer multi-user video conferencing systems, a number of possible types of architectures are introduced $A = \{ A_{\chi}, \chi \in ST \}$, as which the architecture of the client part is distinguished A_{CLIENT} , the architecture of the backend A_{SERVER} and the architecture of data exchange A_{EXCHANGE} . To connect these sets with each other, we introduce a system dynamic alternative multigraph of the following form:

$$A_{\chi}^t = \langle X_{\chi}^t, F_{\chi}^t, Z_{\chi}^t \rangle, \quad (1)$$

where χ — index, characterizing type of video conferencing application architecture, $\chi = \{1, 2, 3\}$ — multiple indexes corresponding to the client, server and data exchange architecture, respectively;
 $t \in T$ - many moments of time;

$X^t = \{x_{\chi}^t, l_{\chi} \in L_{\chi}\}$ - many elements included in the architecture A_{χ}^t at time t ;

$F_{\chi}^t = \{f_{\langle \chi, l, l' \rangle}^t, l, l' \in L\}$ - set of arcs of graph type A_{χ}^t reflecting the relationship between its elements at time t ;

$Z_{\chi}^t = \{f_{\langle \chi, l, l' \rangle}^t, l, l' \in L_{\chi}\}$ - the set of values of the parameters that quantitatively characterize the relationship of the corresponding elements of the graph.

Let's set the set of allowed operations of displaying the above graphs on each other:

$$M^t_{\langle \chi, \chi' \rangle}: Z_{\chi}^t \rightarrow F_{\chi'}^t \quad (2)$$

as well as the operation of the composition of these maps:

$$M^t_{\langle \chi, \chi' \rangle} = M^t_{\langle \chi, \chi_1 \rangle}, M^t_{\langle \chi_1, \chi_2 \rangle}, \dots, M^t_{\langle \chi', \chi' \rangle}$$

then a lot of architectural state can be defined as a subset of the Cartesian product of sets of elements on which the corresponding architecture of the video conferencing application is built:

$$S_{\delta} \subseteq X_1^t \times X_2^t \times X_3^t, \delta = 1, \dots, K_{\Delta}$$

Many many architectural states of a video conferencing application will be written in the following way:

$$S = \{S_{\delta}\} = \{S_1, \dots, S_{K_{\Delta}}\}.$$

We introduce a set of valid operations to map many architectural states of a video conferencing application onto each other:

$$\Pi^t_{\langle \delta, \delta' \rangle}: S_{\delta} \rightarrow S_{\delta'}.$$

In this case we assume that every multi-architecture application state of the video conference is set as a result of the composition operation corresponding graph describing each type of architecture.

All possible data changes are described by action types $Type_A$, which are predetermined by the developer. To use an action, you must call the F_A function of the action which will send a message S to the storage change module M_{SC} . The sent message S must contain the type of action $Type_A$ and a new information I , intended for storage M_{SS} . One of the main algorithms of this module M_{SS} is the data change algorithm in the state store that is implemented in the storage change module M_{SC} . The change module storage M_{SC} provides information processing I from module M_A , and action creates a new state data $State_D$ for warehouse. It is important to note that the M_{SC} storage state change module should create a new *Data* set, but not modify the old data. This limitation is due to the fact that the video conferencing application should store intermediate states that are easy to monitor and debug when

developing and running a video conferencing application. As a result, the storage change module M_{SC} specifies how the $State_A$ of the video conferencing application should change in response to a specific action that occurred in the application.

2.2 The architecture of the interaction of the modules of the web application video conferencing

The architecture $A_{EXCHANGE}$ ($\chi = 3$), presented in figure 2.3, prevents loss of "signal" data when three or more video conferencing participants are connected. Basic structural elements X_3^t of the architecture $A_{EXCHANGE}$ are: 1 – client part of the application; 2 – the server part of the application; 3 - block of data transfer protocols. The client part is divided into two independent components — the user's device and the web page. The user device in the app is required to create audio and video streams from the camera and microphone connected to or being part of the device.

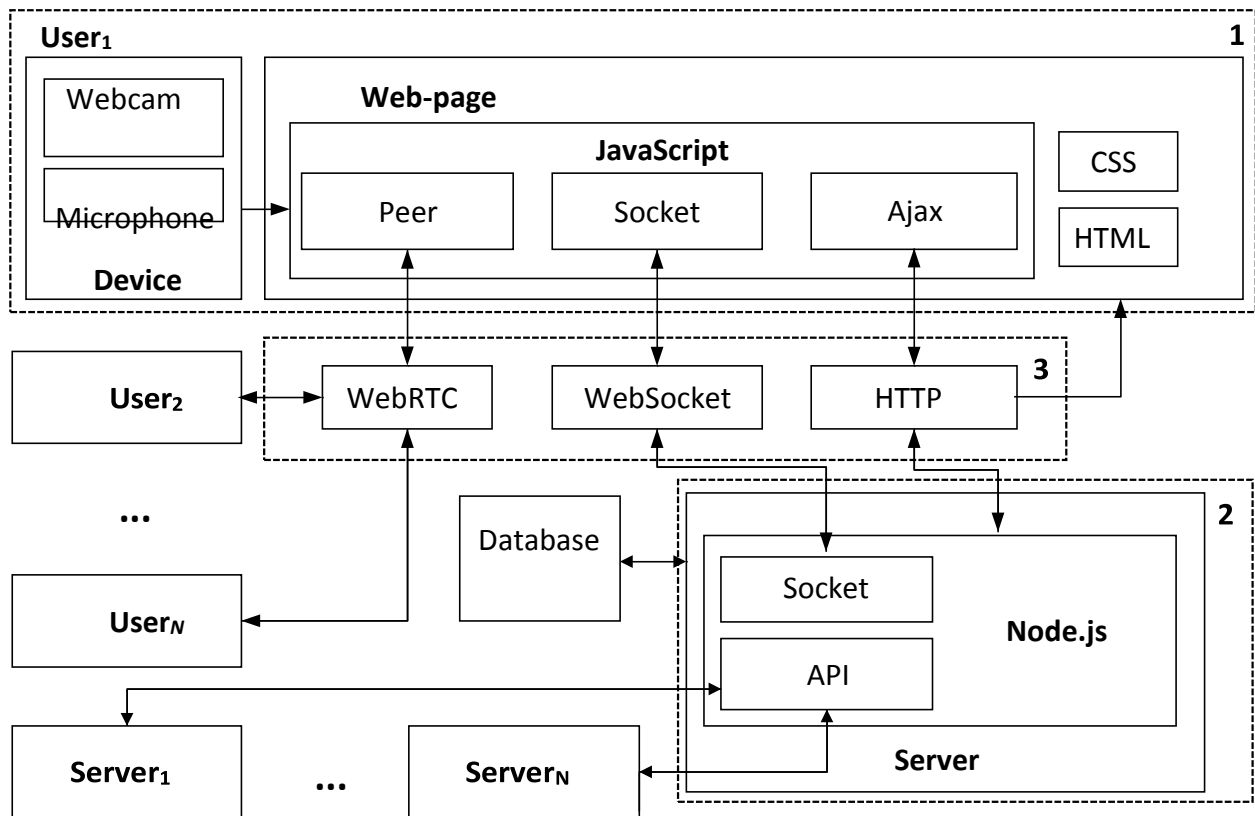


Fig. 2.1. Developed architecture for data exchange in the application of video conferencing

The client side web page of the application consists of classes written in the JavaScript programming language that are required to create connections to the server and other clients through various protocols and data processing. CSS and HTML tools are used to build a graphical interface, display data, and manage the client side of the application. The JavaScript tools used in the video chat web page include three different types of instructions that allow you to organize data transfer over three protocols: WebRTC, WebSocket, and HTTP. JavaScript tools are also used to capture and process data streams from the microphone and video camera.

The next main element of the application architecture is the server part. It performs several different functions: formation of the client part of the application; registration of the client; authorization of the client; exchange of "signal" data between clients; creation of chat rooms and work with the database. The server itself runs on the Node platform.js, which translates JavaScript into machine code and has the same asynchronous architecture as the client side developed by the JavaScript programming language. The MongoDB database, located in the back end of the application, has a no. SQL architecture that is suitable to simplify the implementation of the back end and allows you to quickly adapt its data to changes in the structure of the application. Interaction with the MongoDB database is performed using JavaScript and a special driver library designed for this database.

The third element of the architecture shown in figure 2.1, consists of protocols — HTTP, WebSocket, and WebRTC. These protocols provide data exchange at various stages of application operation, with their help creating connection of client parts via WebRTC Protocol for streaming audio and video data between them. There are problems when creating a connection that originate from the asynchronous application architecture and the WebRTC Protocol that does not provide for the standard implementation of multiple client connections. It is also necessary to note the complexity of the procedure for establishing communication between clients using the WebRTC Protocol, which requires the exchange of "signal" data between them and requires special attention when creating a connection.

In this work, the solution described by the aforementioned problems of loss of signal data with multiple connecting clients' videoconferencing with the introduction of new algorithms of interaction of client and server parts and using various protocols to exchange information. This architecture allows you to create a full peer-to-peer video conferencing application that can work in group video chat mode. The following section describes the main protocols and software tools used to create a client web page and its operation during a video conference.

2.3 Algorithms for establishing connections between clients via WebRTC Protocol

To clearly understand the problem of "signal" loss and proposed in this study software-algorithmic data solutions, first consider the main stages of the functioning of the client and server parts of the developed video conferencing application.

The client part of the application begins with the formation of a web page of registration or authorization, allowing the client to communicate with the server by sending or receiving data via HTTP Protocol.

HTTP is an application layer Protocol for arbitrary data transfer. The Protocol is used in the application to send the client a graphical interface in the form of HTML and CSS data, the logic of the client part of the application written in the programming language JavaScript, as well as to exchange client data with the server when registering and authorizing the client using Ajax technology, which allows you to exchange data with the server.

Customer authorization allows the user to access personal data and the video conferencing page. To authorize the user enters data in the form "login" and "password". Then the data is collected from the forms and sent to the server using Ajax technology. Then the server processes the received data: checks for compliance with a certain set of characters and for exceeding the maximum size of the data in the request, searches for a login-password pair in the database. If all operations are successful, the server will form a video conferencing page with user data and send it to the HTTP client. If the data does not meet certain requirements or if an error occurs, the server will send an information message to the client, which helps to resolve the situation using the HTTP Protocol.

Client registration, as well as authorization, involves filling out forms with data and sending them to the server via HTTP. Then the server processes the data sent by the client. In case of a positive result of processing, the server will store all the data in the database, will automatically register the client, will form and send a response to the client's request in the form of a video conferencing page with user data. In case of incorrect data, the server will return an error notification to the client via HTTP.

Thus, the application uses the HTTP Protocol for reliable transmission of HTML, CSS and JavaScript data between the server and the client. The advantage of using this Protocol is that it is specifically designed to transmit web pages and their logic, and is well supported by all existing browsers. The HTTP Protocol has a set of standard commands, among which there are two main commands: "GET" and "POST", respectively, which allow you to make requests for pages to be issued to the server and a request for exchange of different types of data between the server and the client when authorizing or registering the client.

After the client receives a web page for video conferencing by means of JavaScript, a socket is created on the client, which establishes a connection to the server using the WebSocket Protocol. WebSocket is a full-duplex communication Protocol over a TCP connection designed to exchange messages between the browser and the web server in real time. The WebSocket Protocol opens sockets on the client and server, allowing any type of data to be exchanged. In case of a successful connection via WebSocket Protocol, the server will create a socket with the client data and start its authorization: will try to get http cookie client data that stores the necessary information for authorization, will unpack the cookie data, will try to load from the database session corresponding to the data from the http cookie, on the loaded session will determine the user belonging to the session, bind the user data to the socket, will create a unique number for the socket that will generate and send the "connection" event inside the server. If one of the socket authorization actions generates an error, the socket on the server will be automatically disconnected and removed, and the client socket will receive a connection termination message.

Event "connection" that occurs on the server binds to the socket that is created on the server — the "listeners" of the events that are sent by the client socket. When the client socket is created, it forms a set of "listeners" for events sent by the server socket. Thus, the connection between the client and the server is

established through the WebSocket Protocol, which allows them to quickly exchange messages of various types that do not require their identification, since there is a separate "listener" for each type of message.

This Protocol allows to achieve high speed of information exchange and reduce the load on the client and server due to the absence of costs for identification of data flows. WebSocket Protocol plays an important role in the developed video conferencing application — it is engaged in the transmission of "signal" data of client browsers that allow you to create a connection using the WebRTC Protocol. Thus, this Protocol is the basis for creating a connection using the WebRTC Protocol and simplifies the process of transferring the data necessary for a peering connection.

After establishing a connection with the server via the WebSocket Protocol, the user needs to turn on the video camera and microphone to make video calls and give access to them to the browser. The browser that has access to the camera and microphone of the user, using JavaScript tools, will form media streams of data from connected devices. The received audio and video streams can be transmitted via WebRTC Protocol between the client browsers directly. WebRTC - Internet Protocol designed to organize streaming data between browsers or other applications that support it by point-to-point technology. To connect two clients using WebRTC, you need the following set of JavaScript instructions: create a peer for each of the clients; the appoint of one of its clients as a "calling"; the appoint of another client as "answering"; the formation of a "signaling" data; exchange of signaling data; finishing the connection establishment.

For the transmission of signaling data between clients a server and the WebSocket Protocol are used. Previously created sockets in the client part of the application allow you to transfer "signal" data on certain channels to the server, the server in turn transmits this data to other clients for which they are intended. To connect clients via WebRTC Protocol three types of data are required: "call offer", "call answer" and "candidate". "Call offer" is used to initialize the WebRTC session, it is formed on one of the clients and is sent to the server using WebSocket Protocol, the server in turn sends this message to the "responding" client. "Call offer" is in the format of SDP (Session Description Protocol). An SDP message sent from one node to another that can specify: destination addresses that serve as multicasting media streams, UDP port numbers for the sender and receiver, media formats (such as codecs) that are used during the session, start and stop times. The SDP message is used for broadcast sessions, such as television, radio programs, or video conferences. The client who received the "call offer" will generate and send a response via WebSocket Protocol in the form of "call answer" data, which also have the SDP format. As soon as the client who sent the "call offer" receives the SDP message of "call answer" type, the "candidate" type data will be exchanged between the clients via WebSocket Protocol. Data of type "candidate" has the format ICE (Interactive Connectivity Establishment) Candidate. Creating an interactive connection (ICE) is a method used in computer networks that includes network address transfer (NATs) in Internet applications such as ip telephony (VoIP), peer — to-peer communications (peer-to-peer communications) applications, video

applications, instant messaging (instant messaging) applications, and other interactive media applications. Data type of "candidate" is used for clients' connection, setting the path between them, by which media streams will be transmitted. If the "candidate" type data exchange is successful, each client will open a channel to transfer various types of data via WebRTC Protocol, including audio and video streams.

The WebRTC Protocol has features that create difficulties when connecting users: to create a connection between clients, you need to perform a "handshake" operation, which consists of exchanging different types of "signal" data between browsers, but at the same time the client can establish only one connection using the WebRTC Protocol.

This specificity of the Protocol entails a number of problems in the creation of a full video conferencing. Problems arise due to the asynchronous architecture of the application in connection situations: a single client with a set, a set with one, a set with a set. Such situations result in a violation of the connection algorithm — complete or partial loss of data required to establish communication between clients. To solve this problem, several additional approaches were proposed to build the architecture of data exchange in the application: buffering the "signal" data of the WebRTC Protocol on the client and the server; combining the sockets connected to the clients in the "room" on the server. By doing this a "room" isolates set of sockets from each other. In this way, "rooms" isolate groups of sockets from each other, helping to distribute data only within certain groups. Such approaches help to avoid data loss, create all necessary connections between clients and manage client connection processes at different stages of the application. Next, we consider the algorithms based on the developed approaches that allow you to create a connection using the WebRTC Protocol, to control the processing "signal" data, buffer "signal" data and group client sockets into separate "rooms".

The algorithm shown in figure 2.2 describes the stage of connecting clients before the formation of "signal" data. First, the "calling" client submits a request to the server to connect to the "responding" client via WebSocket Protocol. Next, the server searches for the "responding" client among the connected. If there is no client, the server will end the call of the "calling" client, if "responsible" client was found, he sent the connection request to the WebSocket Protocol. The "responding" client forms and sends a response to the request. If the answer is negative, the server ends the call to the "calling" client. In case of a positive response, the server will receive the socket id of the "calling" and "responding" clients, the socket id will find the "room" in which the sockets are currently located. After this algorithm is completed, socket buffers are created and processed on the server by the algorithm shown in figure 2.3.

After the "room" where the sockets of each client are located is formed, the algorithm presented in figure 2.3 begins to work. First, the socket is extracted from the "room" of the "responding" client, a buffer is created for it to store the sockets waiting for the WebRTC connection. Then, the socket from the "room" of the "responding" client adds to the existing buffer — a socket taken from the "room" of the "calling" client. If the socket taken from the "calling" client's "room" was not

the last one, the operation of extracting the socket from the "calling" client's "room" and adding it to the buffer is repeated with the next socket from this room. After the last socket from the "room" of the "calling" client is taken, the socket from the "room" of the "responding" client is disconnected from its "room" and added to the "calling" client's room.

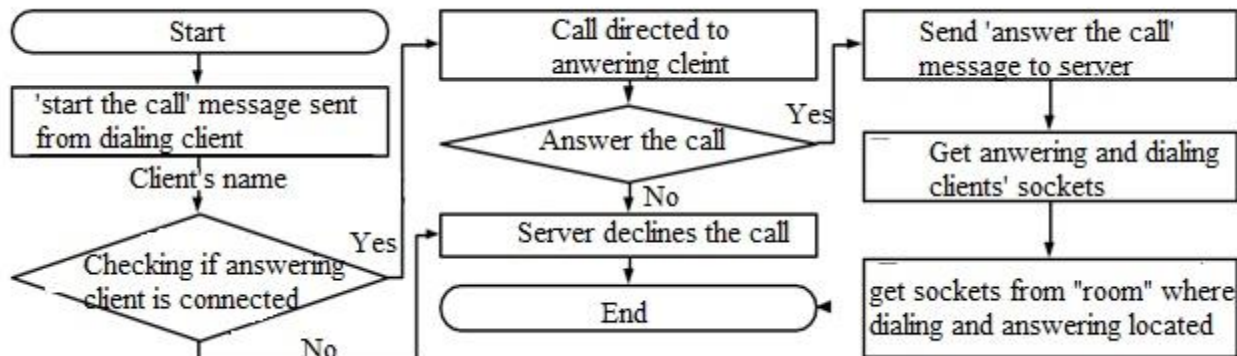


Fig. 2.2. The algorithm of preparation of the client before generating the signal data

Next, the socket buffer processing function is called, which will execute a request to generate signal data for all clients that are in the queue of this buffer. At the end of the algorithm, the "room" of the "responding" client is checked for emptiness. If the "room" is not empty, the algorithm will repeat all actions from the beginning, otherwise the algorithm is considered complete. As you can see, this algorithm adds to the "room" of the "calling" client of the "room" of the "responding" client, and each time you access the "room" of the "calling" client in this algorithm, the "room" should increase. But this is not the case, since before the beginning of the algorithm there is a duplication of all users of the "room," "the calling" client in a separate array. Thus, the algorithm works correctly and every time it uses not the main "room", but a pre-prepared array.

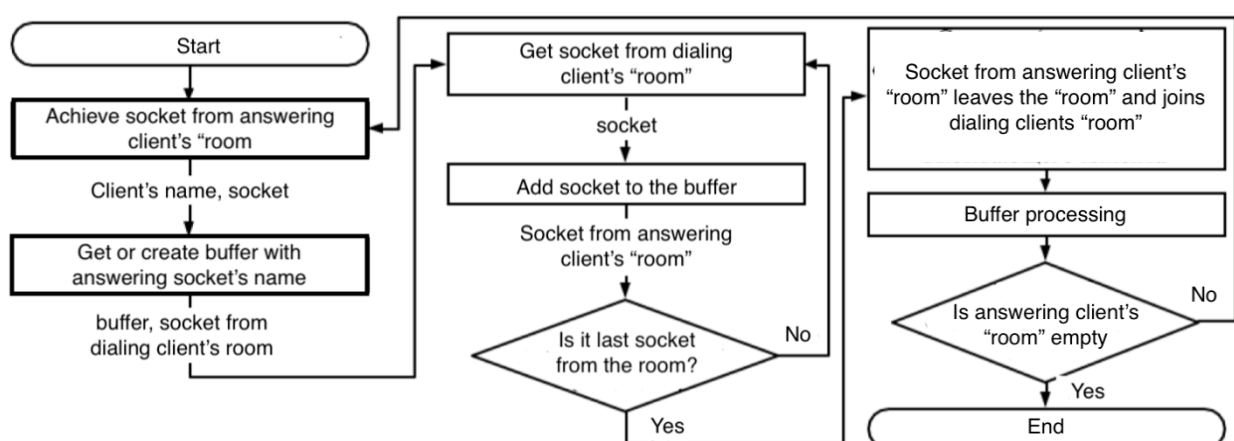


Fig. 2.3. The algorithm for allocating sockets and processing of the buffers on the server

The algorithm, shown in figure 2.4, is shared for processing requests from servers on the formation of data signal "call offer", or for processing received from

another client signal data "call answer". There are two separate buffers for each data type in the video conferencing client application.

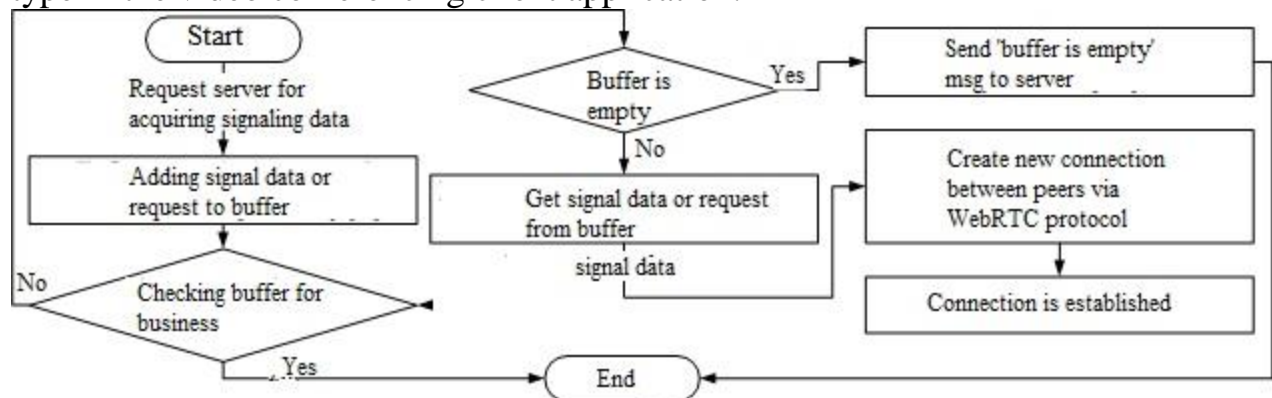


Fig. 2.4. Algorithm for working with the data buffer on the

The incoming request or "signal" data is first added to its corresponding buffer. Then, both buffers are checked for processing one of them at the moment. If one of the buffers is occupied by a processing function, the algorithm terminates and new data in the buffer will be processed later. If none of the buffers are occupied, they are checked for emptiness. If both buffers are empty, the WebSocket client will notify the server that it is ready to receive new data to establish a connection with other clients. If any of the buffers contains data, the first data in the queue will be retrieved and processed accordingly to establish a connection. Once the connection is established, the algorithm continues from the buffer's busy polling location.

It is worth noting that "signal" data of the "candidate" type has no special buffers for storing it either on the client or on the server, as processing occurs immediately as soon as the client receives them. During the processing of "signal" data of type "candidate", buffers belonging to the "call answer" or "call offer" data type, depending on the processing situation, continue to be occupied by the processing function. Thus, the rest of the data type "call answer" or "call offer" continues to be added to the buffers, without breaking the algorithm of client connection via WebRTC Protocol.

The three above algorithms make up one of the main parts of the application, which creates video conferencing with other users via the webrtc peer-to-peer Protocol. The algorithms allow to control asynchronous architecture of client and server parts of the application, processing data as necessary, preventing the occurrence of situations of data mixing and interruption of running connections via WebRTC Protocol. Therefore, the application provides the ability to create group video conferences and monitor their status using the "rooms" clients on the server.

2.4 Acquiring Audio and Video with getUserMedia

The Media Capture and Streams W3C specification defines a set of new JavaScript APIs that enable the application to request audio and video streams from the platform, in addition to a set of APIs to manipulate and process the acquired media streams. The MediaStream object (Figure 2.5) is actually the main interface that enables all of this functionality.

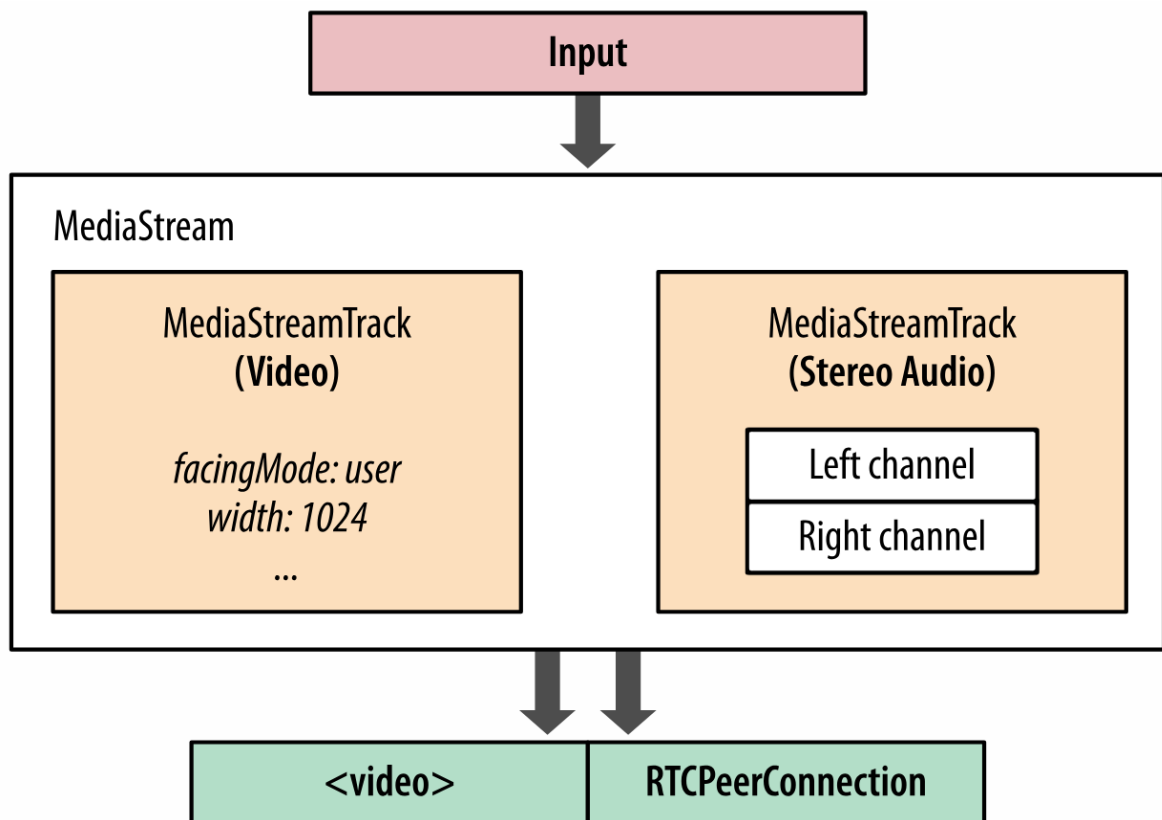


Figure 2.5 – MediaStream carries one or more synchronized tracks

- The MediaStream object consists of one or perhaps more individual tracks (MediaStreamTrack).
- Tracks within a MediaStream object are actually synchronized with each other.
- The input source can be a physical device, like a microphone, webcam or even a remote or local file from the user 's hard drive or perhaps a remote network peer.
- The output of a MediaStream can be delivered to one or even more destinations: a local video or perhaps audio element, JavaScript code for post processing, or perhaps a remote peer.

A MediaStream object represents a real time media stream and allows the application code to acquire data, manipulate individual tracks, and specify outputs. All of the audio and video processing, such as noise cancellation, image enhancement, equalization, and more are easily handled by the audio and video engines.

Nevertheless, the functions of the acquired media stream are actually constrained by the capabilities of the input source: a microphone is able to emit only an audio stream, and some webcams are able to produce higher resolution video streams than others. As a result, when requesting media streams in the browser, the `getUserMedia()` API allows us to specify a list of optional and mandatory constraints (Figure 2.6) to fit the requirements of the application:

```

<video autoplay></video> ❶

<script>
  var constraints = {
    audio: true, ❷
    video: { ❸
      mandatory: { ❹
        width: { min: 320 },
        height: { min: 180 }
      },
      optional: [ ❺
        { width: { max: 1280 } },
        { frameRate: 30 },
        { facingMode: "user" }
      ]
    }
  }

  navigator.getUserMedia(constraints, gotStream, logError); ❻

  function gotStream(stream) { ❼
    var video = document.querySelector('video');
    video.src = window.URL.createObjectURL(stream);
  }

  function logError(error) { ... }
</script>

```

Figure 2.6 – List of constraints

1. HTML video output element
2. Request a mandatory audio track
3. Request a mandatory video track
4. List of mandatory constraints for video track
5. Array of optional constraints for video track
6. Request audio and video streams from the browser
7. Callback function to process acquired MediaStream

This example illustrates among the more elaborate scenarios: we're requesting audio and video tracks, and we're specifying both the minimum resolution and type of camera that has to be used, and a list of optional constraints for 720p Hd video! The `getUserMedia()` API is actually responsible for requesting a chance to access the microphone and camera from the user, and acquiring the streams that match the specified constraints - that is the whirlwind tour.

The provided APIs also enable the application to manipulate individual tracks, modify constraints, clone them, and more. Additionally, once the stream is actually acquired, we are able to feed it right into an assortment of other browser APIs:

- Web Audio API enables processing of audio in the internet browser.
- Canvas API enables post-processing and capture of individual video frames.
- CSS3 and WebGL APIs is able to apply a variety of 2D/3D effects on the output stream.

To make a long story short, `getUserMedia()` is actually a simple API to acquire audio and video streams from the underlying platform. The media is automatically optimized, encoded, and decoded by the WebRTC audio and video engines and it is then routed to one or perhaps more outputs. With that, we're halfway to building a real time conferencing application - we simply have to route the data to a peer!

2.5 Real-Time Network Transport

Real-time communication is time sensitive; that might come as no surprise. As a result, audio and video streaming applications are actually supposed to tolerate intermittent packet loss: the audio and video codecs are able to fill in small data gaps, often with little effect on the output quality. Similarly, applications must implement their own logic to recover from lost or perhaps delayed packets carrying some other types of application data. Timeliness and low latency can be a little more significant compared to reliability.

The necessity for timeliness over reliability is actually the main reason why the UDP protocol is actually a preferred transport for delivery of real time data. TCP delivers a reliable, ordered stream of data: if an intermediate packet is actually lost, then TCP buffers all the packets after it, waits for a retransmission, and then delivers the stream in order to the application.

2.6 RTCPeerConnection API

Regardless of the various protocols involved in setting up and maintaining a peer-to-peer connection, the application API exposed by the browser is fairly easy. The `RTCPeerConnection` interface (Figure 2.7) is actually responsible for managing the total life cycle of each peer-to-peer connection.

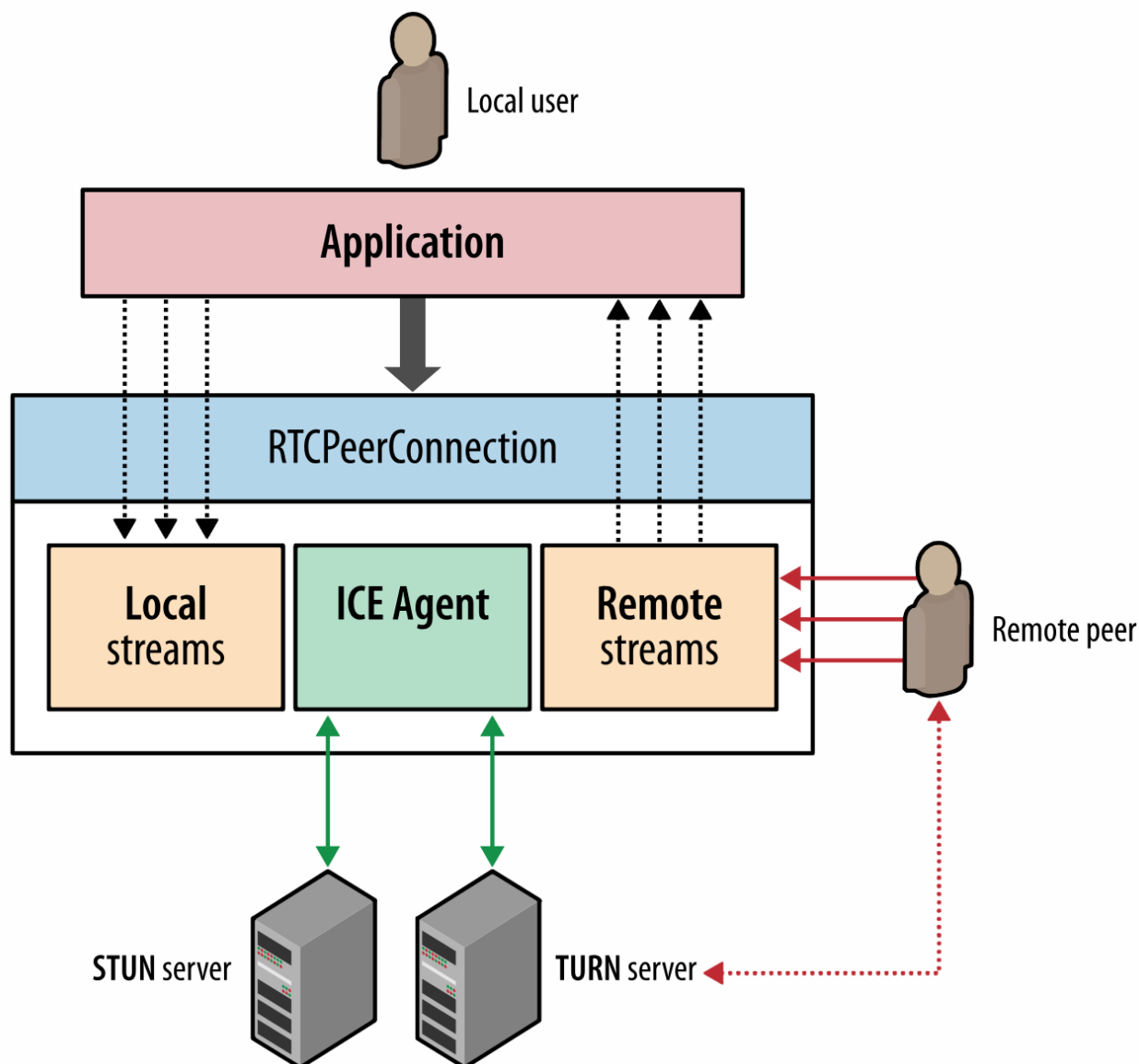


Figure 2.7 – RTCPeerConnection API

- RTCPeerConnection manages the entire ICE workflow for NAT traversal.
- RTCPeerConnection transmits automated (STUN) keepalives among peers.
- RTCPeerConnection keeps an eye on local streams.
- RTCPeerConnection keeps track of remote streams.
- RTCPeerConnection triggers automatic stream renegotiation as needed.
- RTCPeerConnection provides necessary APIs to generate the connection offer, accept the answer, lets us query the connection for the current state of its, and much more.

2.7 Establishing a Peer-to-Peer Connection

Initiating a peer-to-peer connection needs much more work than just opening an XHR, EventSource, or perhaps a new WebSocket session: the latter 3 rely on a well-defined HTTP handshake mechanism to negotiate the parameters of the connection, and all 3 implicitly assume that the destination server is actually reachable by the client - i.e., the server has a publicly routable IP address or perhaps the client and server are actually located on the same internal network.

By comparison, it's very likely that the two WebRTC peers are actually within their own, distinct private networks and behind one or even more levels of NATs. As a result, neither peer is directly reachable by the other. In order to begin a session, we should first gather the possible IP and port candidates for each peer, traverse the NATs, and then run the connectivity checks to find the ones that work, and even then, you will find no guarantees that we'll succeed.

Nevertheless, while NAT traversal is actually an issue we should deal with, we might have gotten ahead of ourselves already. When we open an HTTP connection to a server, there's an implicit assumption that the server is actually listening for the handshake of ours; it may want to decline it, but it's nevertheless always listening for new connections. Unfortunately, the same cannot be said about a remote peer: the peer may be unreachable or offline, busy, or perhaps simply not interested in initiating a connection with the other party.

As a result, in order to build a successful peer-to-peer connection, we must first solve several additional problems:

1. We must notify the other peer of the intent to open a peer-to-peer connection, such it knows to start listening for incoming packets.
2. We must identify potential routing paths for the peer-to-peer connection on each side of the connection and relay the info between peers.
3. We must exchange the required info about the parameters of the various media and data streams - protocols, encodings used, and so on.

The best part is the fact that WebRTC solves one of the issues on our behalf: the built in ICE protocol performs the necessary routing and connectivity checks. Nevertheless, the delivery of notifications (signaling initial session negotiation and) is actually left to the application.

2.7 Multiparty Architectures

One single peer-to-peer connection with bidirectional Hd media streams could easily use up a considerable portion of users' bandwidth. Due to this fact, multiparty applications should carefully think about the architecture (Figure 2.8) of the way in which the individual streams are actually aggregated and distributed between the peers.

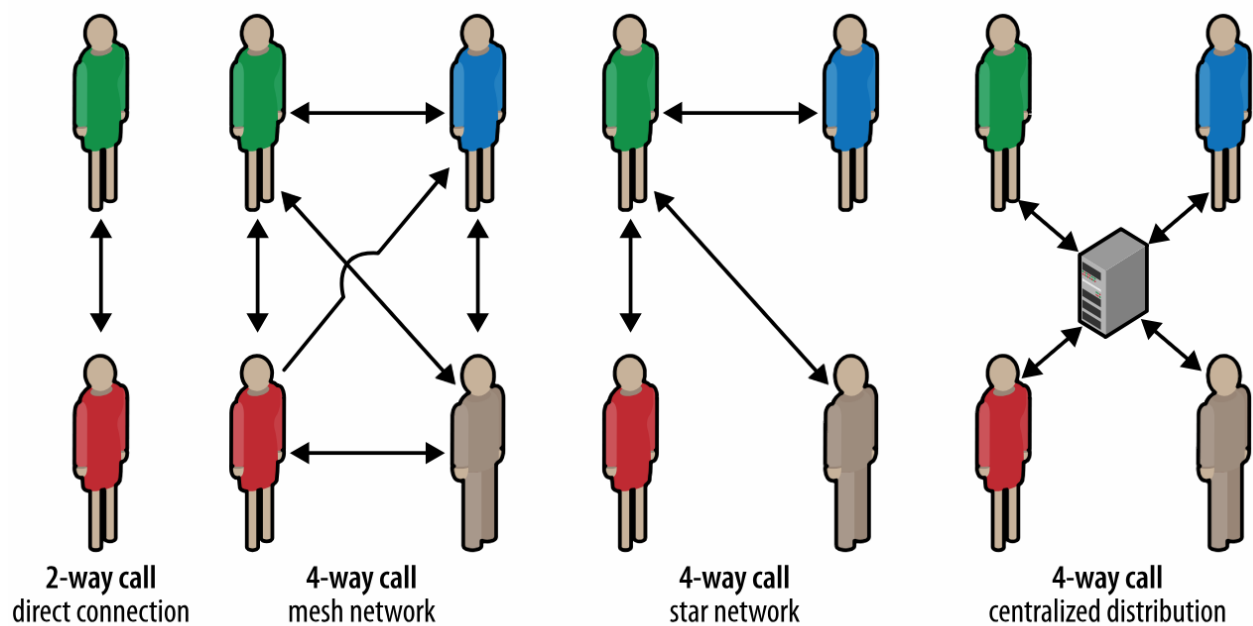


Figure 2.8 – Distribution architecture for an N-way call

One-to-one connections are actually not hard to control and deploy: the peers talk straight to one another and no additional optimization is needed. Nevertheless, extending the identical method to a N way call, where each peer is actually responsible for connecting to every other party (a mesh network) would result in connections for each peer, in addition to a total of connections! If bandwidth is actually at a premium, since it usually is a result of the much lower uplink speeds, then this architecture type will immediately saturate most users' links with only a couple of participants.

While mesh networks are actually not hard to set up, they're usually inefficient for multiparty systems. To manage this, an alternative approach is usually to use a "star" topology instead, where the individual peers connect to a "supernode," which is then accountable for distributing the streams to all connected parties. This way only one peer has to pay the expense of handling and distributing streams, and everybody else talks directly to the supernode.

A supernode can be another peer, or perhaps it is often a dedicated service specially optimized for processing and distributing real time data; which strategy is much more appropriate depends on the application and the context. In probably the simplest case, the initiator is able to act as a supernode - simple, and this may just work. An even better strategy might be to pick the peer with probably the best available throughput, but that also requires additional "election" and signaling mechanisms.

Lastly, the supernode might be a dedicated and also a third-party service. WebRTC allows peer-to-peer communication, but this doesn't mean that there's no room for centralized infrastructure! Individual peers are able to establish peer connections with a proxy server and yet get the advantage of both the WebRTC transport infrastructure as well as the additional services provided by the server.

2.8 Listing of program

```
<title>WebRTC screen share demo</title>
<style>
h1, h2, h3 {
  background: rgb(238, 238, 238);
  border-bottom-width: 1px;
  display: block;
  margin-top: 0;
  padding: .2em;
  text-align: center;
}
.left-video {
  width: 512px;
  height: 384px;
  border: 1px solid black;
}
.right-video {
  width: 512px;
  height: 384px;
  border: 1px solid black;
}
.left-section {
  float: left;
}
.right-section {
  float: right;
}
.buttons-left-section {
  position: absolute;
  float: left;
}
.buttons-right-section {
  position: absolute;
  right: 6px;
}
</style>
```

So this code below refers to css part of the program which answers for screen sharing windows buttons on the bottom of it and overall look of the program in the browser.

Cascading Style Sheets (CSS) is actually a style sheet language used for describing the presentation of a document created in a markup language as HTML. CSS is actually a cornerstone technology of the world Wide Web, alongside JavaScript and HTML.

CSS was created to allow the separation of content and presentation, such as layout, colors, and fonts. This separation is able to improve content accessibility, provide control and flexibility more in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate,css file, and reduce repetition and complexity in the structural content.

```

<h1>Screen share using WebRTC</h1>
<h2>Use Chrome or Firefox, connect two browsers. <br>
Firefox allows screen, window and application share, Chrome only allows screen
share when not an extension.</h2>
<div class="left-section">
<h3>Local Screen</h3>
<video class="left-video" id="localvideo" autoplay controls></video>
<div class="buttons-left-section">
  <button type="button" onclick="startMedia();">Start media</button>
  <button type="button" onclick="stopMedia();">Stop media</button>
  <select id="selector" onchange="selectChanged()" disabled>
    <option value="screen">Screen</option>
    <option value="window">Window</option>
    <option value="application">Application</option>
  </select>
</div>
</div>
<div class="right-section">
<h3>Remote Screen</h3>
<video class="right-video" id="remotevideo" autoplay controls></video>
<div class="buttons-right-section">
  <button type="button" onclick="share();">Share</button>
  <button type="button" onclick="endShare();">End Share</button>
</div>
</div>

```

This part of the code answers for title text on the web page and which function should it operate when user click to button. It gives 3 options of screen sharing: the whole screen, the chosen window or application. Also there is 2 more buttons answering for the start of streaming “share” and its finish “end share”.

```

var hostArray = window.location.host.split(':');
var serverLoc = 'wss://' + hostArray[0] + ':443/'
var socket = new WebSocket(serverLoc);
var localvid = document.getElementById('localvideo');
var remotevid = document.getElementById('remotevideo');
var shareSelector = document.getElementById('selector');
var localStream = null;

```

```

var pc = null;
var mediaFlowing = false;
var useH264 = true;

```

This part of the code answers for server detection from URL.

```

var userAgent = navigator.userAgent.toLowerCase();
var browserM =
userAgent.match(/(opera|chrome|safari|firefox|msie)[\s]*(\d\.[\d]+)/);
var browser = navigator.appName.toLowerCase();
if (browserM)
    browser = browserM[1];
var isChrome = (browser === "chrome");
var isFirefox = (browser === "firefox");

```

This part dedicated to browser detection.

```

var screen_constraints = null;
if (isChrome) {
    screen_constraints = {
        video: {
            mandatory: {
                chromeMediaSource: 'screen',
                maxWidth: screen.width,
                maxHeight: screen.height,
                minFrameRate: 1,
                maxFrameRate: 5
            },
            optional: []
        }
    };
} else {
    selector.disabled = false;
    screen_constraints = {
        video: {
            mediaSource: "screen"
        }
    };
}

```

```

var offerAnswerConstraints = {
    optional: [],
    mandatory: {
        offerToReceiveAudio: true,
        offerToReceiveVideo: true
    }
};

```

This is the constraints of the program. Although this's a web centric world, people's conceptions of design are likely to be framed by print design, where a

billboard is always the same size, a newspaper ad is always the same size, and a magazine cover is actually the same size regardless of who's viewing it, or perhaps exactly where they're reading through the magazine.

Another constraint of site design is actually that unlike print designs, where the viewing area of any design is actually fixed, web users can (and do) zoom in or perhaps out as they interact with a web page, changing the size of images and text. And, by the way, different browsing environments handle zoom differently - some enlarge images as text is actually enlarged, and other times enlarging text does not affect other page elements.

```
function selectChanged() {
  var value = shareSelector.options[shareSelector.selectedIndex].value;
  if (value == "window") {
    screen_constraints = {
      video: {
        mediaSource: "window"
      }
    };
  } else if (value == "screen") {
    screen_constraints = {
      video: {
        mediaSource: "screen"
      }
    };
  } else if (value == "application") {
    screen_constraints = {
      video: {
        mediaSource: "application"
      }
    };
  }
}
```

So this part explains how function of choosing which window the user want to share is working.

```
function startMedia() {
  var promisifiedOldGUM = function(constraints, successCallback,
errorCallback) {
    // First get ahold of getUserMedia, if present
    var getUserMedia = (navigator.getUserMedia ||
      navigator.webkitGetUserMedia ||
      navigator.mozGetUserMedia);
    // Some browsers just don't implement it - return a rejected promise with an
error
```

```

        // to keep a consistent interface
        if(!getUserMedia) {
            return Promise.reject(new Error('getUserMedia is not implemented in
this browser'));
        }
        // Otherwise, wrap the call to the old navigator.getUserMedia with a Promise
        return new Promise(function(successCallback, errorCallback) {
            getUserMedia.call(navigator, constraints, successCallback,
errorCallback);
        });
    }
    // Older browsers might not implement mediaDevices at all, so we set an
empty object first
    if(navigator.mediaDevices === undefined) {
        navigator.mediaDevices = {};
    }
    // Some browsers partially implement mediaDevices. We can't just assign an
object
    // with getUserMedia as it would overwrite existing properties.
    // Here, we will just add the getUserMedia property if it's missing.
    if(navigator.mediaDevices.getUserMedia === undefined) {
        navigator.mediaDevices.getUserMedia = promisifiedOldGUM;
    }
    navigator.mediaDevices.getUserMedia(screen_constraints)
        .then(function(stream) {
            localStream = stream;
            if (localvid.mozSrcObject) {
                localvid.mozSrcObject = stream;
                localvid.play();
            } else {
                try {
                    localvid.src = window.URL.createObjectURL(stream);
                    localvid.play();
                } catch(e) {
                    console.log("Error setting video src: ", e);
                }
            }
        })
        .catch(function(err) {
            console.log(err.name + ": " + err.message);
            if (location.protocol === 'http:') {
                alert('Please test this WebRTC experiment on HTTPS.');
```

```

    }
    console.error(e);
  });
}
function onerror(e) {
  if (location.protocol === 'http:') {
    alert('Please test using HTTPS.');
```

enabled the appropriate flag? see README.md');

```

  } else {
    alert('Screen capturing is either denied or not supported. Have you
    }
    console.error(e);
  }
}
```

So this is the main part of the program, in which we can clearly see that how WebRTC protocol involved into it by using `getUserMedia` function. The `MediaStream` interface represents a stream of media content. A stream consists of several tracks such as video or perhaps audio tracks. Each track is actually specified as an instance of `MediaStreamTrack`. You is able to obtain a `MediaStream` object either by making use of the constructor or perhaps by calling `MediaDevices.getUserMedia()`.

Some user agents subclass this interface to provide more accurate info or perhaps functionality, like in `CanvasCaptureMediaStream`.

Each `MediaStream` has an input, which might be a `MediaStream` generated by `navigator.getUserMedia()`, and an output, which might be passed to a video element or an `RTCPeerConnection`.

```

function stopMedia() {
  localvid.src = "";
  localStream.getVideoTracks()[0].stop();
}
function useH264Codec(sdp) {
  var isFirefox = typeof InstallTrigger !== 'undefined';
  if (isFirefox)
    updated_sdp = sdp.replace("m=video 9 UDP/TLS/RTP/SAVPF 120 126
97\r\n", "m=video 9 UDP/TLS/RTP/SAVPF 126 120 97\r\n");
  else
    updated_sdp = sdp.replace("m=video 9 UDP/TLS/RTP/SAVPF 100 101
107 116 117 96 97 99 98\r\n", "m=video 9 UDP/TLS/RTP/SAVPF 107 101 100 116
117 96 97 99 98\r\n");
  return updated_sdp;
}
}
```

This code above answers for terminating the local video stream or screen sharing.


```

function setLocalDescAndSendMessageOffer(sessionDescription) {
  if (useH264) {
    // use H264 video codec in offer every time
    sessionDescription.sdp = useH264Codec(sessionDescription.sdp);
  }
  pc.setLocalDescription(sessionDescription);
  console.log("Sending: SDP");
  console.log(sessionDescription);
  socket.send(JSON.stringify({
    "messageType": "offer",
    "peerDescription": sessionDescription
  }));
}

```

In this part of code SDP packets are sent over web socket.

```

function setLocalDescAndSendMessageAnswer(sessionDescription) {

  if (useH264) {
    // use H264 video codec in offer every time
    sessionDescription.sdp = useH264Codec(sessionDescription.sdp);
  }

  pc.setLocalDescription(sessionDescription);
  console.log("Sending: SDP");
  console.log(sessionDescription);

  socket.send(JSON.stringify({
    "messageType": "answer",
    "peerDescription": sessionDescription
  }));
}

function onCreateOfferFailed() {
  console.log("Create Offer failed");
}

```

This is the continuation of sending SDP packets over web socket via WebRTC protocol.

```

function share() {
  if (!mediaFlowing && localStream) {
    createPeerConnection();
    mediaFlowing = true;
  }
}

```

```

        pc.createOffer(setLocalDescAndSendMessageOffer,
onCreateOfferFailed, offerAnswerConstraints);
    } else {
        alert("Local stream not running yet or media still flowing");
    }
}

```

As we can understand in this section of code there is a function which turns on when you press the ‘start share’ button.

```

function endShare() {
    console.log("end share");
    socket.send(JSON.stringify({type: "bye"}));
    stop();
}
function stop() {
    if (pc) {
        pc.close();
    }
    pc = null;
    remotevid.src = null;
    mediaFlowing = false;
}
function onCreateAnswerFailed(error) {
    console.log("Create Answer failed: ", error);
}
socket.addEventListener("message", onWebSocketMessage, false);

```

And this section of code is vice-versa answers for ‘end share’ button.

```

function onWebSocketMessage(evt) {
    var message = JSON.parse(evt.data);
    if (message.messageType === 'offer') {
        console.log("Received offer...")
        if (!mediaFlowing) {
            createPeerConnection();
            mediaFlowing = true;
        }
        console.log('Creating remote session description...');
        var remoteDescription = message.peerDescription;
        var RTCSessionDescription = window.RTCSessionDescription ||
window.webkitRTCSessionDescription || window.RTCSessionDescription;
        pc.setRemoteDescription(new
RTCSessionDescription(remoteDescription), function() {
            console.log('Sending answer...');

```

```

        pc.createAnswer(setLocalDescAndSendMessageAnswer,
onCreateAnswerFailed);
    }, function() {
        console.log('Error setting remote description');
    });
} else if (message.messageType === 'answer' && mediaFlowing) {
    console.log('Received answer...');
    console.log('Setting remote session description...');
    var remoteDescription = message.peerDescription;
    var RTCSessionDescription = window.RTCSessionDescription ||
window.webkitRTCSessionDescription || window.RTCSessionDescription;
    pc.setRemoteDescription(new
RTCSessionDescription(remoteDescription));
} else if (message.messageType === 'iceCandidate' && mediaFlowing) {
    console.log('Received ICE candidate...');
    var RTCIceCandidate = window.RTCIceCandidate ||
window.webkitRTCIceCandidate || window.RTCIceCandidate;
    var candidate = new
RTCIceCandidate({ sdpMLineIndex:message.candidate.sdpMLineIndex,
sdpMid:message.candidate.sdpMid, candidate:message.candidate.candidate });
    console.log(candidate);
    pc.addIceCandidate(candidate);

} else if (message.type === 'bye' && mediaFlowing) {
    console.log("Received bye");
    stop();
}
}
function createPeerConnection() {
    console.log("Creating peer connection");
    RTCPeerConnection = window.webkitRTCPeerConnection ||
window.RTCPeerConnection;
    var pc_config = {"iceServers":[]};
    try {
        pc = new RTCPeerConnection(pc_config);
    } catch (e) {
        console.log("Failed to create PeerConnection, exception: " + e.message);
    }
    // send any ice candidates to the other peer
    pc.onicecandidate = function (evt) {
        if (evt.candidate) {
            console.log('Sending ICE candidate...');
            console.log(evt.candidate);

            socket.send(JSON.stringify({

```

```

        "messageType": "iceCandidate",
        "candidate": evt.candidate
    }));

    } else {
        console.log("End of candidates.");
    }
};
console.log('Adding local stream...');
pc.addStream(localStream);
pc.addEventListener("addstream", onRemoteStreamAdded, false);
pc.addEventListener("removestream", onRemoteStreamRemoved, false)
// when remote adds a stream, hand it on to the local video element
function onRemoteStreamAdded(event) {
    console.log("Added remote stream");
    remotevid.src = window.URL.createObjectURL(event.stream);
}

// when remote removes a stream, remove it from the local video element
function onRemoteStreamRemoved(event) {
    console.log("Remove remote stream");
    remotevid.src = "";
}
}

```

All this big section of the code is taking care of process messages that are coming from web socket during our WebRTC peer-to-peer screen sharing session.

Mentioned above part of the code was related to WebRTC session itself. But that's not all. We have another part of the program binded to Javascript programming language. And it is responsible for what is happening in the *node.js* part of the program. *Node.js* is the server part.

```

var WebSocketServer = require('websocket').server;
var https = require('https');
var fs = require('fs');
var clients = [];

var options = {
    key: fs.readFileSync('webrtcwebsocket-key.pem'),
    cert: fs.readFileSync('webrtcwebsocket-cert.pem'),
};

var server = https.createServer(options, function(request, response) {
    fs.readFile(__dirname + '/index.html',
        function (err, data) {

```

```

        if (err) {
            response.writeHead(500);
            return response.end('Error loading index.html');
        }
        response.writeHead(200);
        response.end(data);
    });
});

server.listen(443, function() {
    console.log((new Date()) + " Server is listening on port 443");
});

// create the server
wsServer = new WebSocketServer({
    httpServer: server
});

function sendCallback(err) {
    if (err) console.error("send() error: " + err);
}

// This callback function is called every time someone
// tries to connect to the WebSocket server
wsServer.on('request', function(request) {
    console.log((new Date()) + ' Connection from origin ' + request.origin +
'.');

    var connection = request.accept(null, request.origin);
    console.log(' Connection ' + connection.remoteAddress);
    clients.push(connection);

    // This is the most important callback for us, we'll handle
    // all messages from users here.
    connection.on('message', function(message) {
        if (message.type === 'utf8') {
            // process WebSocket message
            console.log((new Date()) + ' Received Message ' + message.utf8Data);
            // broadcast message to all connected clients
            clients.forEach(function (outputConnection) {
                if (outputConnection !== connection) {
                    outputConnection.send(message.utf8Data, sendCallback);
                }
            });
        }
    });
});
});

```

```

connection.on('close', function(connection) {
    // close user connection
    console.log((new Date()) + " Peer disconnected.");
});
});

```

3. Life safety section

3.1 Assessment of the forthcoming physical and mental load at service of the technical characteristic of the equipment of the workplace of the service personnel

This diploma project considered the development of applications for the generation of business plans. Development is made on one laptop. In the room under consideration is 1 employee-computer operator.

The working environment of the PC operator is a set of physical, chemical, biological, socio-psychological and aesthetic environmental factors affecting the operator.

A comprehensive assessment of the working environment factors is carried out on the basis of the method of physiological classification of the severity and intensity of work.

Table 3.1 — Categories of severity of work

Index of categories of severity of work	Characteristic category of severity of work
I	<p>Activities, the presence of which the impact of unsafe and harmful conditions leads to the development of the most complete neighboring capital in almost strong people. Most of the physical conditions of this presence is exacerbated, especially at the end of the employee stages (replacement, weeks). There are typical manufacturing predefined capital diazaborine, etc.</p> <p>Work, the presence of which in consequence of extremely negative circumstances of work at the end of the labor stage (replacement, weeks) are created by the interaction, characteristic for the purpose of painful multifunctional capital of the body in almost strong people, disappearing from many employees after full entertainment. But certain people have all the chances to switch to production and high-class diseases.</p>
II	<p>Work performed in particularly negative (dangerous) circumstances of work. The presence of this painful interaction is formed very rapidly, have all chances to have an irreparable appearance and are often accompanied by serious violations of the functions of</p>

	important organizations. Activities carried out in circumstances where the maximum permissible concentration (CONCENTRATION) and the maximum permissible degree (PDU) of harmful and unsafe working conditions does not exceed the conditions of regulatory industrial papers.
III	The presence of this functionality does not break, declensions in staying well-being, interfaced with high-class activism, does not appear throughout the whole stage of the person's work.
IV	The presence of this functionality does not break, declensions in staying well-being, interfaced with high-class activism, does not appear throughout the whole stage of the person's work.
V	Activities, the presence of which the impact of unsafe and dangerous harmful conditions leads to the development of the most complete neighboring capital in almost strong people. Most of the physical conditions of this presence is exacerbated, especially at the end of the employee stages (replacement, weeks). There are characteristic production predetermined capital of pre-illness, etc.

The study of high-grade notch is carried out taking into account the magnitude of the properties of health and loss of ability to work.

Factors of region PC operator close up to the optimization of factors:

The temperature of the atmosphere in the RM in the room in the years warmed. In the production rooms, in which the activities of the computer and VT is considered the main, must be guaranteed rational characteristics of the local climate.

The temperature of the atmosphere is 20C.

The duration of the exposure conditions — 480 minutes.

On this basis 2 points are displayed.

The specific gravity of the impact of the condition in the length of the replacement proletarians can be thought out according to the composition:

$$t = 480/480 = 1. \quad (3.1)$$

The specific assessment of the indicator according to the formula 1 will be:

$$X_4 = 2 \times 1 = 2. \quad (3.2)$$

Table 3.2 – Working environment factors

Working environment factors	Indicator	Value of indicator	Score factor after optimization	The duration of the factor t _{pm} after optimization	assessment of the specific gravity of the working environment

					factor $X\phi$
The temperature of the air in the workplace, °C:					
warm period	X1	21-22	2	480	2
cold period	X2	20-22	1	480	1
Industrial dust, the multiplicity of exceeding the MPC, times.	X3	-	1	480	1
Vibration, exceeding the MPL, dB	X4	lower than MPL	1	480	1
Industrial noise, exceeding the MPL, dB	X5	< 1	1	480	1
Ultrasound, exceeding the MPL, dB	X6	< 1	1	480	1
Heat radiation intensity, W/m ²	X7		1	480	1
Illumination of the workplace, LC:	X8	at the level of sanitary norms	1	480	1
min object of distinction, mm	X9	> 1	1	480	1
category of work	X10	3-4	1	480	1
Physical dynamic load, j:					
total $\times 10^5$	X11	4,2	1	480	1
regional $\times 10^5$	X12	2,1	1	480	1
Physical static load, N · s:					
on one hand $\times 10^4$	X13	< 18	1	480	1
on both hands $\times 10^4$	X14	< 43	1	480	1
on body	X15	< 61	1	480	1

muscles x10 ⁴					
Workplace (WP), posture and movement in space	X16	RM stationary, posture free, weight of the transported cargo up to 5 kg	1	480	1
Shift	X17	morning shift	1	480	1
Duration of continuous work during the day, h	X18	4	1	480	1
Duration of concentrated observation, % of the length of the work shift	X19	51-57	3	96	0,6
Number of important monitoring objects	X20	< 5	1	480	1
Tempo (number of movements per hour):					
small (wrist)	X21	361-720	2	480	2
large (hands)	X22	< 250	1	480	1
The number of calls in an hour	X23	76-175	2	480	2
Monotony:					
the number of receptions of the operation	X24	6-10	2	480	2
the duration of the recurring transactions, s	X25	31-100	2	480	2
Work and rest mode	X26	with the inclusion of music and	1	480	1

		gymnastics			
Nervous-emotional load	X27	Complex actions on a given plan with the possibility of correction	3	240	1,5

Integrated scoring severity of labor is determined by the formula 3.3:

$$U_T = X_{\max} + (6 - X_{\max})/6(N-1)Xf_i, \quad (3.3)$$

where X_{\max} - the highest of the obtained partial points;

Xf_i – score on i-th of the factors taken into account;

n – number of factors taken into account without one factor X_{\max} ;

N – total number of factors.

In this case, the values will look like:

$$X_{\phi i} = 35,5;$$

$$U_T = 3,661.$$

Table 3.3- Integral score

Category of severity	1	2	3	4	5	6
Integral-point estimation	Up to 1.8	1,9-3,3	3,4-4,5	4,6-5,3	5,4-5,9	6,0 and more

The result of the accumulated score is considered to be the group of seriousness of works 3.

The main problem is considered to be self-optimization of the meaning of the conditions of the labor site of the operator for the purpose of more effective its activity. Analyzing the data earlier, in order to optimize the circumstances of the work, it is necessary to reduce the period of irritable and psychological overload, up to 240 min., to reduce the period of constant activity during the days.

3.2 The calculation of the integral scoring after optimization

It is necessary to calculate the integral score after optimization.

Integrated scoring severity of labor is determined by the formula 5.3.

In this case, the values will look like:

Table 3.3 - Working environment factors

Working environment	Indicator	Value of indicator	Score factor after	The duration of	assessment of the
---------------------	-----------	--------------------	--------------------	-----------------	-------------------

factors			optimization	the factor tpmin after optimization	specific gravity of the working environment factor Xφ
The temperature of the air in the workplace, °C:					
warm period	X1	21-22	2	480	2
cold period	X2	20-22	1	480	1
Industrial dust, the multiplicity of exceeding the MPC, times.	X3	-	1	480	1
Vibration, exceeding the MPL, dB	X4	lower than MPL	1	480	1
Industrial noise, exceeding the MPL, dB	X5	< 1	1	480	1
Ultrasound, exceeding the MPL, dB	X6	< 1	1	480	1
Heat radiation intensity, W/m ²	X7		1	480	1
Illumination of the workplace, LC:	X8	at the level of sanitary norms	1	480	1
min object of distinction, mm	X9	> 1	1	480	1
category of work	X10	3-4	1	480	1
Physical dynamic load, j:					
total x10 ⁵	X11	4,2	1	480	1
regional x10 ⁵	X12	2,1	1	480	1
Physical static load, N · s:					
on one hand	X13	< 18	1	480	1

x10 ⁴					
on both hands x10 ⁴	X14	< 43	1	480	1
on body muscles x10 ⁴	X15	< 61	1	480	1
Workplace (WP), posture and movement in space	X16	RM stationary, posture free, weight of the transported cargo up to 5 kg	1	480	1
Shift	X17	morning shift	1	480	1
Duration of continuous work during the day, h	X18	4	1	480	1
Duration of concentrated observation, % of the length of the work shift	X19	51-57	3	96	0,6
Number of important monitoring objects	X20	< 5	1	480	1
Tempo (number of movements per hour):					
small (wrist)	X21	361-720	2	480	2
large (hands)	X22	< 250	1	480	1
The number of calls in an hour	X23	76-175	2	480	2
Monotony:					
the number of receptions of the operation	X24	6-10	2	480	2
the duration of the recurring transactions, s	X25	31-100	2	480	2

Work and rest mode	X26	with the inclusion of music and gymnastics	1	480	1
Nervous-emotional load	X27	Complex actions on a given plan with the possibility of correction	3	240	1,5

$$X_{\phi i} = 32,1;$$

$$U_T = 2,098.$$

Table 3.4 - Integral score

Category of severity	1	2	3	4	5	6
Integral-point estimation	Up to 1.8	1,9-3,3	3,4-4,5	4,6-5,3	5,4-5,9	6,0 and more

3.3 Conclusion

Prior to the optimization, i.e. in the category of severity of work equal to 3, some production indicators were reduced-optimization of work and rest. In such circumstances, the position of the body of a reasonable worker reached up to the last, thus, what increases the likelihood of diseases and different signs in workers. In order to optimize the circumstances of the work, it is necessary to reduce the period of irritable and psychological overload, up to 240 min., to reduce the period of continuous activity during the days, to reduce the duration of the careful study up to 20%. After the optimization of the characteristics of the group of the seriousness of the same 2. The presence of this group shall be observed without exception of the permissible significance of the conditions of the proletarian sphere. The presence of this functionality does not break, deviations in the stay of health, associated with high-class work cannot be traced.

4. Economical part

4.1 Calculation of the cost of software development

According to the formula 4.1, the number of costs required for SOFTWARE development is determined, which, in turn, includes expenses, accruals, depreciation and labor payment:

$$C = WF + SST + D + EC + CMC + MC + OE + OH, \quad (4.1)$$

where: WF – wage fund;
 SST – deductions for social security tax;
 D – depreciation;
 EC – electricity costs;
 CMC – costs of materials and components;
 MC – the cost of maintenance;
 OE – other expenses;
 OH – overhead.

Two components of the salary are the main salary and additional. The wage Fund is calculated by adding the basic and additional wages according to the following formula:

$$WF = S_b + S_{add}, \quad (4.2)$$

where: S_b – basic salary, thousand tenge;
 S_{add} – additional salary, thousand tenge.

The basic salary is considered according to the formula:

$$S_b = T \times TC / (t_{cp} \times 8), \quad (4.3) \text{ where}$$

t_{cp} – the average number of days in a month is 21 days, multiplied by the number of hours in a work day – 8;

TC – tariff rate.

4.2 Calculation of the complexity of software development

To calculate the components of labor costs is determined by the formula 5.4:

$$Q = q \times c, \quad (4.4)$$

where Q – conditional number of commands;

q – coefficient that takes into account the conditional number of commands depending on the type of task;

c – coefficient taking into account the novelty and complexity of the program.

In this thesis we used multivariate problems, and the coefficient, which takes into account the conditional number of teams is 5200.

Next you need to determine the C-coefficient, which takes into account the novelty and complexity of the program.

According to the degree of novelty, the software can be divided into 4 groups:

- group A - development of fundamentally new tasks;
- group B - development of original programs;
- group C - development of programs using standard solutions;
- group D - one-time typical task.

The coefficient of calculation of labor intensity is selected from table 5.1, on the cross-arrangement of groups of complexity and the degree of novelty.

Table 4.1 – The payoff of the complexity

Computer language	Difficulty group	Degree of novelty				Coefficient C
		A	B	C	D	
High level	1	1,38	1,26	1,15	0,69	1,2
	2	1,30	1,19	1,08	0,65	1,35
	3	1,20	1,10	1,00	0,60	1,5
Low level	1	1,58	1,45	1,32	0,79	1,2
	2	1,49	1,37	1,24	0,74	1,35
	3	1,38	1,26	1,15	0,69	1,5

For the diploma project was chosen to develop a fundamentally new program in a high-level programming language with 2 levels of complexity. So $C = 1.35$.

Next, it is necessary to calculate the main indicator of the component parts of labour costs according to the formula 4.1.

$$Q = 5200 \times 1,35 = 7020 \text{ (commands).}$$

Then you need to calculate the time to develop the software. The total time for product development consists of different components.

The composition of the full time to design the application is shown in 4.2.

Table 4.2 – The structure of the total time to create a software product

Stage	Indication of the time of this stage	The Maintenance of stage
1	Tpd	Preparation of the description of task
2	Td	Problem description
3	Ta	Algorithm development
4	Tfc	Develop a flowchart of the algorithm
5	Tw	Writing a program in Javascript
6	Tp	Program printing
7	Tot	Debugging and testing the program
8	Tn	Documentation, user instructions, explanatory note

In man-hours the time is calculated, while Tpd is taken from the actual time, the time of the remaining parts is characterized by the calculated method, according to the conditional number of commands Q.

The time that was spent on one stage of product design can be calculated by the formula:

T_{pd} (time for the preparation of the description of the problem), characterized by the fact and is (from 3 to 5 days 8 hours):

$$T_{pd} = 24 \text{ person / h.}$$

T_d (time for the description of the problem) is calculated by the formula:

$$T_d = Q \times C / (50 \times K), \quad (4.5)$$

where C – the coefficient of accounting for changes in the problem, the coefficient C depending on the complexity of the problem and the number of changes is selected in the range from 1.2 to 1.5. In this paper, the coefficient C = 1.35, selected from table 4.1.

K – coefficient, taking into account the qualification of the programmer.

Determine the value of the coefficient K from the table 4.3

Table 4.3 – The coefficients of the skill of the programmer

Experience	The coefficient of skill
till 2 years	0,8
2-3 years	1
3-5 years	1,1 – 1,2
5-7 years	1,3 – 1,4
more than 7 years	1,5 – 1,6

In the thesis qualification coefficient K = 0,8, as the experience of working activity of the programmer is not more than 2 years.

The value of Q was calculated by the formula 5.4, Q = 7020. The calculation of the (time for the description of the problem) is calculated by the formula 4.5:

$$T_d = 7020 \times 1,35 / (50 \times 0,8) = 236,9 \text{ hours.}$$

T_a (time to design the algorithm) determined by the formula 4.6:

$$T_a = Q / (50 \times K), \quad (4.6)$$

The value Q = 7020, the qualification coefficient K = 0.8 was taken from table 5. Calculation of T_a is calculated by the formula 5.6:

$$T_a = 7020 / (50 \times 0,8) = 175,5 \text{ hours.}$$

T_{fc} (the time for the preparation of the flowchart) is also calculated by the formula T_a 4.6.

$$T_{fc} = 7020 / (50 \times 0,8) = 175,5 \text{ hours.}$$

T_w (time of writing the product in the programming language) is considered according to the formula 4.7:

$$T_w = Q \times 1,5 / (50 \times K). \quad (4.7)$$

Value of $Q = 7020$, $K = 0.8$, are taken from the table 4.3.

$$T_w = 7020 \times 1,5 / (50 \times 0,8) = 263,25 \text{ hours.}$$

T_p (time printing programs) is considered by the formula 4.8:

$$T_p = Q / 50. \quad (4.8)$$

Value of $Q = 7020$.

$$T_p = 7020 / 50 = 140,4 \text{ hours.}$$

T_{ot} (time of compilation and testing of the program) is considered according to the formula 4.9:

$$T_{ot} = Q \times 4,2/50 \times K. \quad (4.9)$$

The values of Q and K are also calculated in sub-paragraph 4.5.

$$T_{ot} = 7020 \times 4,2 / 50 \times 0,8 = 737,1 \text{ hours.}$$

T_n (time for documentation), taken in fact and equals (from 3 to 5 days 8 hours):

$$T_d = 24 \text{ person / h.}$$

Total labor costs are defined as the total compound cost of the formula 4.10:

$$T = T_{pd} + T_d + T_a + T_{fc} + T_w + T_p + T_{ot} + T_n. \quad (4.10)$$

$$T = 24 + 236,9 + 175,5 + 175,5 + 263,25 + 140,4 + 737,1 + 24 = 1776,65 \text{ hours.}$$

The minimum wage (minimum wage), which from 01.01.2017 in the Republic of Kazakhstan is equal to 24459 tenge, which increases depending on the tariff coefficient that corresponds to this type of work, is the tariff rate.

$$S_b = 1776,65 \times 24459 / (21 \times 8) = 258661 \text{ tg.}$$

Additional salary is equal to 21% of the basic salary, determined by the formula 4.11:

$$S_{add} = 0,21 \times S_b \quad (4.11)$$

$$S_{add} = 0,21 \times 258661 = 54319 \text{ tg.}$$

The total salary is calculated according to the formula 5.2:

$$WF = 258661 \text{ tg} + 54319 \text{ tg} = 312980 \text{ tg.}$$

The social tax is 11% (ст. 358 п. 1 HK PK) from the employee's income, and is determined by the formula 5.12:

$$SST = (WF - ST) \times 11\%, \quad (4.12)$$

where ST – pension contributions, which are equal to 10% of WF and social tax are not taken into account:

$$ST = WF \times 10\% \quad (4.13)$$

ST according to the formula 5.13 is equal to:

$$ST = 312980 \text{ tg} \times 0,1 = 31298 \text{ tg.}$$

Social tax according to the formula 5.12 will be:

$$SST = (312980 \text{ tg} - 31298 \text{ tg}) \times 0,11 = 30985 \text{ tg.}$$

According to the mandatory existing depreciation rates, depreciation is indicated as a percentage of the cost of equipment and is considered according to the formula 4.14:

$$D = \frac{C_{обор} \times H_a \times N}{100 \times 12 \times t}, \quad (4.14)$$

where H_a – depreciation rate;

$C_{обор}$ – initial cost of equipment;

N – time to use your personal computer;

t – number of working days per month.

The cost of the equipment is 160 000 tg.

Depreciation rate (H_a) of the equipment is determined by the formula:

$$H_a = \frac{C_{обор} - C_{ликв}}{T_{норм} \times C_{обор}} \times 100\%, \quad (4.15)$$

where $C_{\text{ликв}}$ — liquidation value, equal to 5.6% of the cost of equipment;
 $T_{\text{норм}}$ — standard service life (for personal computer — 4 years).

$$H_A = \frac{160000 - 8960}{4 \times 120000} \times 100 = 31,5 \text{ .}$$

Then it is necessary to mark N-time of equipment use. The total time of operation of the laptop counts only the time of work on the computer and is determined by the formula 4.16:

$$T = T_a + T_{fc} + T_w + T_p + T_{ot}, \quad (4.16)$$

All values are already known.

$$T = 175,5 + 175,5 + 263,25 + 140,4 + 737,1 = 1492 \text{ hours.}$$

The operating time of the laptop is used in the formula 4.15 in days, so the value that was calculated by the formula 4.16 (hours) is translated into days.

$$1492 \text{ hours} / 8 = 187 \text{ days.}$$

The calculation of depreciation is calculated by the formula 4.14.

$$A = \frac{160000 \times 31,5 \times 187}{100 \times 12 \times 21} = 37400 \text{ руб.}$$

Electricity costs are calculated according to the formula:

$$EC = M \times k_3 \times T \times C_{kWh}, \quad (4.17)$$

where M – power of computer (450 Wt);

k_3 – load factor (0.8);

C_{kWh} – the cost of 1 kWh of electricity;

T - operating time, hour.

$$EC = 0,45 \times 0,8 \times 1492 \times 16,65 = 8943,048 \text{ руб.}$$

The costs of materials and components used in the design of the software application (CMC), as well as the costs of maintenance and repair (MC) will be 1.3% and 2.7% of the cost of equipment – formula (4.18-4.19):

$$CMC = 0,013 \times C_{\text{обор}}, \quad (4.18)$$

$$CCMC = 0,013 \times 160\,000 \text{ руб} = 2080 \text{ руб},$$

$$MC = 0,027 \times C_{\text{обор}}, \quad (4.19)$$

$$MC = 0,027 \times 160\,000 \text{ руб} = 4320 \text{ руб}.$$

The overhead costs associated with the management and maintenance, maintenance and operation of the equipment and other additional costs to

support the processes and circulation are 61.5% of the Wage Fund, determined by the formula 4.20:

$$C_{oh} = 0,615 \times WF, \quad (4.20)$$

$$C_{oh} = 0,615 \times 312980 \text{ tg} = 192483 \text{tg}$$

The General results of the software product cost calculation and its composition are shown in table 4.4 and figure 4.1.

Table 5.4 – The effective cost table of the software application

Item of expenditure	The sum, thousand tenge	Percentage of total
Wages, WF	312980	53,120
Social tax, SST	30985	5,259
Depreciation rate, D	37400	6,348
Electricity costs, EC	8943,048	1,518
Materials and components, CMC	2080	0,353
Overhead cost, Coh	192483	32,669
Maintenance and repair cost, MC	4320	0,733
Total:	589191	100

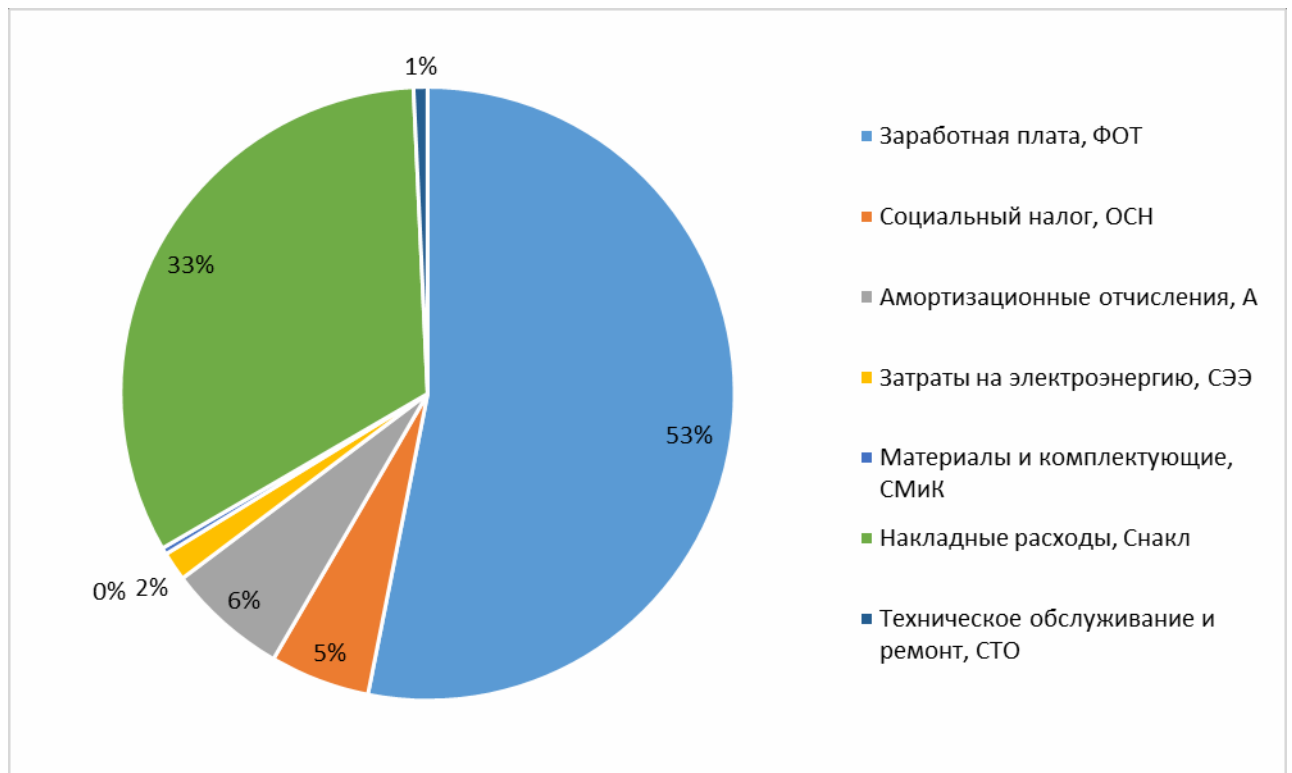


Figure 4.1 - Cost structure of software development

Conclusion

The total cost of developing the software product is 589191 tenge. Since it is intellectual work, the most part is the salary of the developer 312980 tg. The payback period of the project does not exceed year and half.

Conclusion

In this diploma work, WebRTC protocol in peer-to-peer systems was analyzed.

In the first part of the project analysis of the current state of the WebRTC protocol, the prospects for its development in the world of telecommunications was done.

In the second part of the diploma work, we analyzed WebRTC network architecture and technologies used in WebRTC such as WebSocket and algorithms that occur when 2 clients are connected.

In the section of life safety, an analysis of working and microclimate conditions was carried out. The calculation of natural and artificial lighting of the working room is presented. The proposed architecture of data exchange in the video conferencing application allows you to distribute the data channels on the relevant protocols and divide tasks between them. Since initially, the application architecture of video conferencing is asynchronous, was carried out the development and implementation of new algorithms for data management that is required to connect client applications on WebRTC Protocol. As a result, we received a screensharing application that is able to monitor the status of clients connected "rooms» - on the server during online chat.

In the economic part, the business plan of the project is also compiled and a description of the economic efficiency of the project is provided to calculate the payback period and it is equal to 4 months.

Developed peer-to-peer "serverless" architecture and algorithms of interaction of the modules of the video conferencing system connects the client and server parts of the system under the Protocol WebRTC and communication of client web applications.

It is worth noting that at the moment there are some other problems that limit the use of the WebRTC Protocol on devices:

- 1) only three browsers (Opera, Mozilla Firefox, Google Chrome) support this Protocol;

- 2) requires a powerful processor and enough memory to process audio and video data streams. In addition, these browsers do not work with graphics coprocessors, as a result of which the main processor is loaded.

List of abbreviations

1. RTC – Real Time Communication
2. P2P – Peer-to-peer
3. OS – Operating System
4. API –Application Programming Interface
5. JSON – JavaScript Object Notation
6. JS – Javascript
7. HTML –HyperText Markup Language
8. IP – Internet Protocol
9. CPU – Central Processing Unit
10. TCP - Transmission Control Protocol
- 11.URL – Uniform Resource Locator
- 12.HTTP – HyperText Transfer Protocol
- 13.IRC – Internet Relay Chat
- 14.IM – Instant Message
- 15.PeerCDN – Peer-top-peer content delivery
- 16.HD – High Definition
- 17.TB – Terabyte measure of computer storage capacity
- 18.CSS - Cascading Style Sheets
- 19.SDP – Session Description Protocol
- 20.UDP – User Datagram Protocol
- 21.ICE – Interactive Connectivity Establishment
- 22.NAT – Network Address Translation
- 23.STUN – Session Traversal Utilities for NAT
- 24.TURN – Traversal Using Relays around NAT
- 25.PC – Personal Computer
- 26.WF – wage fund;
- 27.SST – deductions for social security tax;
- 28.D – depreciation;
- 29.EC – electricity costs;
- 30.CMC – costs of materials and components;
- 31.MC – the cost of maintenance;
- 32.OE – other expenses;
- 33.OH – overhead

List of references

- 1 Ronzhin, A.L., Saveliev, A.I., Budkov, V.Yu. Context-Aware Mobile Applications for Communication in Intelligent Environment / A.L. Ronzhin, 2 A.I. Saveliev, V.Yu. Budkov // Internet of Things, Smart Spaces, and Next Generation Networking. — Springer Berlin Heidelberg, 2012. — С. 307–315.
- 3 Chan, T. et al. Studying with the cloud: the use of online Web-based resources to augment a traditional study group format / T. Chan, S. Sennik, A. Zaki, B. Trotter // CJEM. — 2014. — Т. 16. — С.34–37.
- 4 Gerpott, T. J., Meinert, P. The impact of mobile Internet usage on mobile voice calling behavior: A two-level analysis of residential mobile communications customers in Germany / T. J. Gerpott, P. Meinert // Telecommunications Policy. — 2016. — Т. 40. — № 1. — С. 62–76.
- 5 Айдынбай, Т. Ж., Шуйтенов, Г. Ж. Технологии передачи данных в системах видеоконференцсвязи / Т. Ж. Айдынбай, Г. Ж. Шуйтенов // Наука, техника и образование. — 2015. — № 4(10). — С.77–83.
- 6 Месяцев С. Мобильная конференцсвязь будущего / С. Месяцев // Мобильные системы. — 2007. — № 12. — С. 48–53.
- 7 H.R. Oh et al. An effective mesh-pull-based P2P video streaming system using Fountain codes with variable symbol sizes / H. R. Oh, D. O.Wu, H. Song // Computer Networks. — 2011. — Т. 55. — С. 2746–2759.
- 8 Губарев, В. В., Обейдат, А. А. Алгоритм взаимного исключения одновременного доступа пользователей к общим ресурсам в пиринговых системах / В. В.Губарев, А. А. Обейдат // Вестник НГТУ. — 2009. — № 2. — С. 75–90.
- 9 Civanlar, M. R. et al. Peer-to-peer multipoint videoconferencing on the Internet / M. R. Civanlar, Ö. Özkasap, T. Çelebi // Signal Processing: Image Communication. — 2005. — Т. 20. — pp.743–754.
- 10 Ramzan, N. et al. Video streaming over P2P networks: Challenges and opportunities / N. Ramzan, H. Park, E. Izquierdo // Signal Processing: Image Communication. — 2012 — Т. 27. — С. 401–411.
- 11 Е.Хакимжанов. Расчет аспирационных систем. Дипломное проектирование. Для студентов всех форм обучения всех специальностей. – Алматы: АИЭС, 2002 - 30 стр
- 12 Абдимуратов Ж.С., Мананбаева С.Е. Безопасность жизнедеятельности: Методические указания к выполнению раздела «Расчет производственного освещения» в выпускных работах для всех специальностей. Бакалавриат - Алматы: АИЭС, 2009. - 20 с
- 13 Базылов К.Б., Алибаева С.А., Бабич А.А. : Методические указания по выполнению экономического раздела выпускной работы бакалавров для студентов всех форм обучения специальности 050719 – Радиотехника, электроника и телекоммуникации – Алматы: АИЭС, - 2008. -19 с.

14 Кодекс Республики Казахстан от 10 декабря 2016 года № 99-IV «О налогах и других обязательных платежах в бюджет (Налоговый кодекс)» (с изменениями и дополнениями по состоянию на 28.04.2016 г.), ст.120.

15 <https://hpbn.co/webrtc/>

Appendix A – List of programming of CSS part of the application

```
<style>
.....
h1, h2, h3 {
    background: rgb(238, 238, 238);
    border-bottom-width: 1px;
    display: block;
    margin-top: 0;
    padding: .2em;
    text-align: center;
}
.left-video {
    width: 512px;
    height: 384px;
    border: 1px solid black;
}
.right-video {
    width: 512px;
    height: 384px;
    border: 1px solid black;
}
.left-section {
    float: left;
}
.right-section {
    float: right;
}
.buttons-left-section {
    position: absolute;
    float: left;
}
.buttons-right-section {
    position: absolute;
    right: 6px;
}
</style>
.....
```

Appendix B – List of programming of server part

```
71 // detect server from URL
72 var hostArray = window.location.host.split(':');
73 var serverLoc = 'wss://' + hostArray[0] + ':443/'
74 var socket = new WebSocket(serverLoc);
75
76 var localvid = document.getElementById('localvideo');
77 var remotevid = document.getElementById('remotevideo');
78 var shareSelector = document.getElementById('selector');
79 var localStream = null;
80 var pc = null;
81 var mediaFlowing = false;
82 var useH264 = true;
83
84 // Detect browser
85 var userAgent = navigator.userAgent.toLowerCase();
86 var browserM = userAgent.match(/(opera|chrome|safari|firefox|msie) [\s\S]*([\d\.]+)/);
87 var browser = navigator.appName.toLowerCase();
88 if (browserM)
89     browser = browserM[1];
90 var isChrome = (browser === "chrome");
91 var isFirefox = (browser === "firefox");
92
93 var screen_constraints = null;
94
95 if (isChrome) {
96     screen_constraints = {
97         video: {
98             mandatory: {
99                 chromeMediaSource: 'screen',
100                 maxWidth: screen.width,
101                 maxHeight: screen.height,
102                 minFrameRate: 1,
103                 maxFrameRate: 5
104             },
105             optional: []
```

Appendix C – List of programming of getUserMedia function

```
176     if(navigator.mediaDevices.getUserMedia === undefined) {
177         navigator.mediaDevices.getUserMedia = promisifiedOldGUM;
178     }
179
180     navigator.mediaDevices.getUserMedia(screen_constraints)
181         .then(function(stream) {
182             localStream = stream;
183             if (localvid.mozSrcObject) {
184                 localvid.mozSrcObject = stream;
185                 localvid.play();
186             } else {
187                 try {
188                     localvid.src = window.URL.createObjectURL(stream);
189                     localvid.play();
190                 } catch(e) {
191                     console.log("Error setting video src: ", e);
192                 }
193             }
194         })
195         .catch(function(err) {
196             console.log(err.name + ": " + err.message);
197             if (location.protocol === 'http:') {
198                 alert('Please test this WebRTC experiment on HTTPS.');
```

```
199             } else {
200                 alert('Screen capturing is either denied or not supported. Have you enabled the appropriate flag? see README.md');
201             }
202             console.error(e);
203         });
204     }
205
206     function onerror(e) {
207         if (location.protocol === 'http:') {
208             alert('Please test using HTTPS.');
```

```
209         } else {
210             alert('Screen capturing is either denied or not supported. Have you enabled the appropriate flag? see README.md');
211         }
212         console.error(e);
213     }
```