HELSINKI METROPOLIA UNIVERSITY OF APPLIED SCIENCES

Master's Degree in Information Technology

Multimedia Communications

Master's Thesis

Web Services as Near the Real Time Transport Protocol for Multimedia Systems

Author: Imran Razzaq
Instructor: Jouko Kurki, DSc (Tech)

Approved: __. __. 2012

PREFACE

This master’s thesis is the result of 3 years of studies and 2 years of research at the Helsinki Metropolia University of Applied Sciences under the supervision of Jouko Kurki.

In the beginning it was difficult for me to choose the research topic and define the scope of the study but a course offered in the first semester “Research Methodologies” helped me with this issue. First of all, I would like to thank Jouko Kurki who helped me to select the research topic, define the research questions associated to it, define the scope of the study and eventually conclude the thesis.

Secondly, I would like to thank Ville Jääskeläinen, Marjatta Huhta and all the other teachers and students at Metropolia who put their effort and time to review the paper and helped me to finalize it. I would also like to thank Jonita Martelius for providing valuable assistance in completing the manuscript.

Thirdly, I would like to say a few words of thanks to my employer SAP and colleagues who helped me to manage time for the Master’s Degree in general and the thesis in particular.

Fourth, I would like to thank my family members, wife, son, brothers, sister and other relatives whose patience was the key for the success of this thesis.

Last, but definitely not the least, I would like to thank to my parents who raised me up, molded me into the man I am today and for being there for me at every stage of my life.

Helsinki, May 11, 2012

Imran Razzaq

ABSTRACT

Name: Imran Razzaq
Title: Web Services as Near the Real Time Transport Protocol for Multimedia Systems
Date: May 11, 2012 **Number of pages:** 58
Degree Programme: Master of Engineering in Information Technology (Multimedia Communications)
Instructor: Jouko Kurki, DSc (Tech)

Web Services in general offer a complete package of functioning services via a loosely coupled distributed communication model for the client applications to access their functions. The architecture of Web Services is mainly based on Internet standards, e.g. TCP/IP, HTTP, SOAP and XML.

Web Services are commonly used in a Client/Server communication model; on the other hand, in the modern telecommunications world the P2P communications have an important role for distributed computing. For instance different servers need to communicate with each other for distributed computing. The present study is about the application and use of the Web Services Platform in Peer to Peer (P2P) architecture, e.g. multimedia systems, by presenting a communication model for P2P applications.

Real time data requires that its accuracy and validity is tightly or loosely coupled with a time frame, which means that the data becomes unused or invalid if not available within a desired time period. If the time period is not very short and strictly followed it is called Near the Real Time Data. Generally web services are used for the Non Real Time data but here the web service platform was experimented to carry Near the Real Time Data by working as a transport layer for Real time Transport Protocols, i.e. RTP, and evaluated in terms of technical feasibility, communication overheads and communication delays.

An experimental system that carries voice data over the Internet was implemented by using the communication model presented. A G.729 voice codec was used to make 128 kb/s coded voice from a sound device of a Windows computer. The voice data was carried over an IP-network using a Web Services based architecture. The SOAP-protocol was used as the transport protocol for Real Time Protocol (RTP) carrying the voice data. The communication overhead was evaluated to be reasonable in a typical network scenario depending upon the network type and the voice data parameters, i.e. bit rate and frame rate. In today’s network conditions communication overhead is acceptable and the proposed architecture of using a Web Services based architecture provides a unified and easy to deploy system for transmission of business critical data in a wide variety of networks.

Key words: Web Services, P2P, Client Server, TCP/IP, HTTP, SOAP, XML, RTP

Table of Contents

1 INTRODUCTION

Web Services offer a complete package of functioning services via a loosely coupled distributed communication model for the client applications to access their functions, e.g. a stock exchange service publishing stocks data over the internet. Web services take the HTTP or web platform to the next level by offering complex functional services over the web. Applications use web services to access the functionality offered by the service over the web. In modern distributed computing systems, large business enterprises have many different dedicated business services (as web series) offered internally or externally and these services need to interoperate to implement a complete business process. The Service-oriented architecture (SOA) is the methodology for achieving applications and services interoperability. In service oriented architecture the web services are used as Peer to Peer communication protocol where the infrastructure is heterogeneous across operating systems, applications, system software, and application infrastructure. In a web services context Peer to Peer communications means that web services are interoperating with other web services and some web services are dedicated to serving clients while other web services are dedicated to serving other web services. [3]

Being an open standard for system interoperability, Web Services have become the widely used standard in multimedia communication systems as well. Multimedia systems are different from other data and business systems e.g. financial systems etc. in a way that multimedia systems involve several different types of data communications, e.g. signaling, real time data transfer etc. In multimedia communication systems, media (e.g. voice) itself is usually communicated over real time which means that the data need to be communicated within a defined time period and latency. The protocol used to communicate such type of data is called Real-time Transport Protocol. In this research project a communication model for such type of multimedia communications is defined.

Web services have been the biggest advantage of Web 2.0 in recent years. Web services are becoming popular for the P2P architecture for the integration of different multimedia subsystems in a service oriented architecture where the web services communicate with each other. So far web services are only popular in web domain but in future web services will be expanded to different other areas as well, e.g. embedded systems and real time systems. In order to support such domains, web services should be able to handle complex data. In this research study a web services platform is explored and experimented for different kind of data - different in type and behavior as well, e.g. real time data that is binary data in its nature and time critical in its behavior. As discussed earlier, the real time data is an essential part of multimedia systems and there is definitely a need for research and analysis for the web services platform to test its capability of being a transport protocol for real time data. In other words, the web services need to be analyzed as a transport layer for real time transport protocols.

1.1 Goals and Research Question

This study aims to explore the web services architecture to support the peer-to-peer communication model and define the communication model for real time media such as voice and explore the possibility of using web services as transport layer for such media; in other terms, using web services as a transport layer for the RTP protocol. The aim of the study is to enable the SOA (service oriented architecture) to expand to real time communications as well.

Following are the main research questions:

1. Does the current technological infrastructure of services oriented architecture allow communicating real time data? How is the binary data to be encoded in the SOAP Protocol and how is the RTP protocol to be presented in the XML format?
2. How much does the SOAP and XML formatting cause communication overhead? Is that acceptable over today's networks available?
3. How efficient is the web based communication for real time data, how much delays are caused because of HTTP/XML/SOAP formatting and encodings? Can the real time effectiveness of data be maintained?

The study also analyzes the latency and bandwidth and different other communication parameters for real time media. Thirdly, the study aims to analyze the benefits of web services for the RTP communication protocol**.**

1.2 Outcome

The following are the outcomes expected from the research:.

- Web Services Peer to Peer Communication Model
- WSDL based description language for real time communications
- Prototype implementation for RTP encapsulations over web services capable of transporting voice data

- Communication overhead and transmission efficiency calculations

The web services peer to peer communication model and web service description language represented here can be used as a guideline to implement such web services which carry real time data in a service oriented architecture.

1.3 Research Methodology

Technical feasibility study, experimental approach and data analysis are used to prove the real time compatibility of the Web Services platform in a peer to peer architecture. The result is in the form of a working prototype communicating for example real time data between two peers i.e. voice data using the Real Time Transport Protocol. Further proceeding with the research, the implemented prototype is used to analyze the data communication to calculate the communication overhead, verify real time effectiveness and efficiency.

2 HISTORY OF WORLD WIDE WEB

World Wide Web was officially introduced to the outside world in 1991 as a communication medium to share knowledge over internet. Later its usage was expanded to commercial use for advertisement and services. The following briefly introduces the two eras of the World Wide Web.

2.1 Birth of Internet and World Wide Web

The World Wide Web is like an encyclopedia, a telephone directory, a related collection, a video shop, and Speaker's Corner all rolled into one and accessible through any computer. Initially web was designed to transfer hyper text (HTML) over the internet. World Wide Web was mainly treated as online store for textual and graphical information, videos, audios etc. Web is typically based on client/server architecture; the system that hosts the website consisting of interlinked web pages (HTML pages) is called web server and the client software that connects with web server to access the HTML pages. The client and server communicate with each other over HTTP protocol that uses TCP/IP as transport layer. The Internet is like a network of electronic roads crisscrossing the planet – the much-hyped information superhighway. The Web is just one of many services using that network, just as many different kind of vehicles use the road. [1]

2.2 Web 2.0

The O'Reilly Media, during the years 2002-2004, introduced with Web 2.0, the 2nd generation of web, as a platform for offering applications and services. E.g. Blogs, Social Networking sites, wikis, RSS, video sharing and online collaboration. In web 2.0 the web site becomes interactive and allows the client applications to feed the website interactively. In 2004, O'Reilly Media and Media Live hosted the first Web 2.0 conference where they represented the Web 2.0 as Web as Platform [2]. The main driving aspects of Web 2.0 can be detailed as follows; *Media Richness*, in Web 2.0 the web applications are developed with multimedia items using the advanced technologies e.g. AJAX. Service Oriented Architecture, Web 2.0 also represents the web as a most suitable and efficient platform for offering a service or a set of services (see Chapter 1.2). Web Services also add the interactivity benefit for the applications. Interactivity: Web 2.0 represents the web not only a read only static/dynamic data resource centres but makes the websites as interactive. Users can own the data on a Web 2.0 site and exercise control over that data. These sites may have an "Architecture of participation" that encourages users to add value to the application as they use it [2].

In summary, the Web 2.0 enabled the web with the power of media richness, service oriented architecture and interactivity to enable the next generation of web as a platform for offering applications and services.

3 WEB SERVICES AND SERVICE ORIENTED ARCHITECTURE

Web services offer a complete package of functioning services via a loosely coupled distributed communication model for the client applications to access their functions. Web Services take the HTTP/web platform to the next level for offering complex functional services over the web. Applications use web services to access the functionality offered by the service over the web [3]. The platform elements of Web Services include XML/SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery and Integration) and WSDL (Web Services Description Language).

Web services are commonly used in a client/server model where there is a server offering the service and clients accessing the web service hosted on the server. The platform expansion of a web service to a network of web services is called SOA (service oriented architecture). In the earlier text, the Web 2.0 was represented as a few new ideas in the web technology; Web Services is one of the most important features of Web 2.0. In this research, the Web 2.0 is addressed from the web services point of view.

3.1 Web Service as Peer to Peer Communication Model

Web services can also be used as a communication model for peer to peer architecture, where different software components communicate with each other using the web service architecture.

E.g. in one multimedia communication system, there is one subsystem that is responsible for switching the voice calls and relay the media, while another subsystem is responsible for recording the media. The recording subsystem needs to get the service of the switching system to record a voice call; on the other hand the switching system needs to get the service of the recording system to record the calls. Both systems can operate as a Client/Server to each other. In this case the communication model between these two systems is called a P2P model and this model can adopt web service as a communication protocol by allowing the web service to web service communications. Figure 1 below describes the above example and how a service oriented architecture works in an enterprise to provide a set of services.

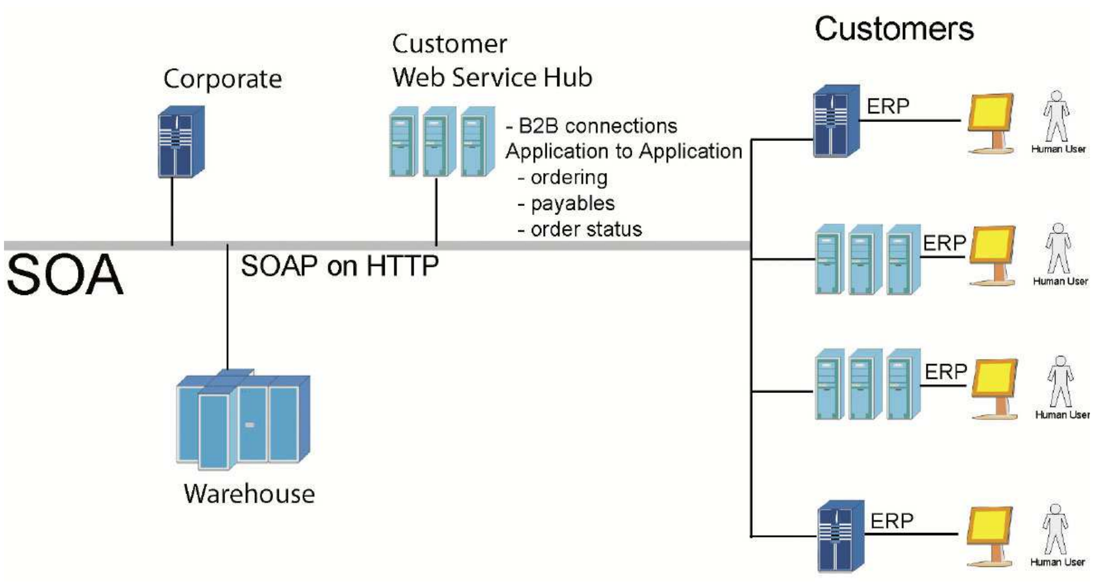


Figure 1: Example of Service Oriented Architecture

Figure 1 shows that different ERP systems act as a web service to its client workstations and that at the same time Warehouse and Customer Web Service Hub acts as a server to the ERP systems and all they share is the common SOA platform.

3.2 Anatomy of Web Service

A web service is a package of functional services available over the Internet and whose functions (requests, responses and events) are offered in the form of SOAP Messages and accessible over the HTTP transport layer. In order to understand a web service it is necessary to understand the building blocks of a web service.

The SOAP messages are the xml encoded messages which travel over the internet through HTTP protocol which uses TCP/IP as transport.

Figure 2 illustrates how Web Services interact with each other in Service Oriented Architecture.

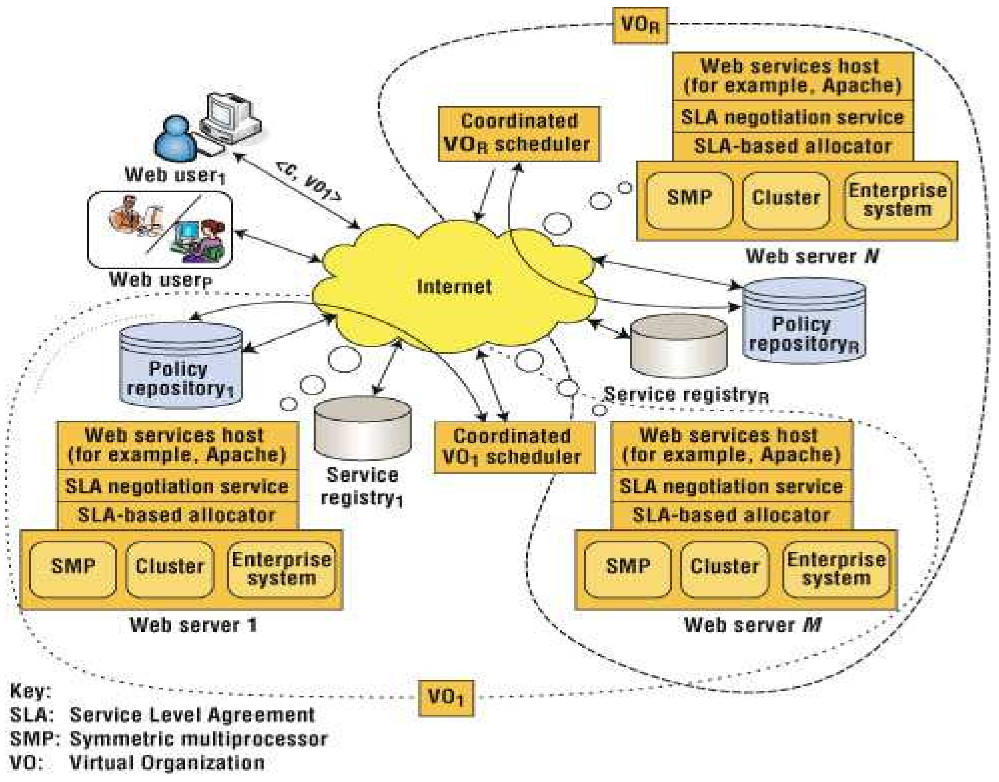


Figure 2: Web Service architecture

Figure 2 shows the architecture of Web Services and a services oriented architecture, which shows how services interface with each other and also with the end user. The main communication backbone is the web with a SOAP/HTTP protocol for the web service to web service interfacing as well as end user to web services interfacing.

3.2.1 HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP protocol involves Requests and Responses to access a resource, create a new resource or modify an existing resource remotely. HTTP has been in use by the World-Wide Web global information initiative since 1990. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1.0, as defined by RFC 1945, improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta information about the data transferred and modifiers on the request/response semantics. Using this feature HTTP protocol extends its usability for complex data types. However, HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or virtual hosts. In addition, the proliferation of incompletely-implemented applications calling themselves "HTTP/1.0" has necessitated a protocol version change in order for two communicating applications to determine each other's true capabilities. [4]

In the web services context, HTTP is used to transport SOAP objects where Get and Post requests carry the SOAP object that is xml encoded data.

3.2.2 XML

XML Stands for Extensible Markup Language, it is a markup Language used for describing information as an electronic document and it stores the information intelligibly; that is why it is called a Meta markup language.

The W3C World Wide Web Consortium is the organization that is maintaining the XML Standard and current specification of the XML is available at www.w3c.org XML exists because HTML was successful. Therefore, XML incorporates many successful features of HTML. XML also exists because HTML could not live up to new demands. [5]

Some of the areas where XML will be useful in the near-term include: Large Web site maintenance. XML would work behind the scene to simplify the creation of HTML documents, exchange of information between organizations, offloading and reloading of databases, syndicated content, where content is being made available to different Web sites, electronic commerce applications where different organizations collaborate to serve a customer, scientific applications with new markup languages for mathematical and chemical formulas, electronic books with new markup languages to express rights and ownership and handheld devices and smart phones with new markup languages optimized for these “alternative” devices.

3.2.3 SOAP

SOAP stands for simple object access protocol; it is a protocol that specifies the Web Services object in the XML encoded data and allows them to be exchanged using HTTP transport protocol. SOAP can express the web services requests as well as responses. SOAP is capable of representing whole web service functions in the form of communicated objects.

A SOAP message is an ordinary XML document containing the following elements: An Envelope element that identifies the XML document as a SOAP message, A Header element that contains header information, A Body element that contains call and response information and a Fault element containing errors and status information.

All the elements above are declared in the default namespace for the SOAP envelope: http://www.w3.org/2001/12/soap-envelope and the default namespace for SOAP encoding and data types is: http://www.w3.org/2001/12/soap-encoding [6]

3.2.4 WSDL

WSDL stands for Web Service Description Language; that is an XML based language that is used to functionally describe a web service. WSDL is a document written in XML. The document describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes. [7]

A WSDL document describes a web service using these major elements:

Element	Defines
<types>	The data types used by the web service
<message>	The messages used by the web service
<portType>	The operations performed by the web service
<binding>	The communication protocols used by the web service

A WSDL document is just a simple XML document. It contains a set of definitions to describe a web service.

Having now discussed the building blocks of Web Services and Service Oriented Architecture, the study moves on to introduce the Real Time data, its communications and how the Web Services Platform could be used as transport protocol for such data.

4 REAL TIME DATA, ITS COMMUNICATIONS AND REAL TIME TRANSPORT PROTOCOL

‘This chapter introduces the real time data, its nature, protocols used for communication and important factors involved in such communications. Real time data is a type of data that is continuous in its nature and its validity or effectiveness is in bind with a time period that is, it is valid within a certain period of time. Following are the main properties of real time data. Reliability: The data must be reliable, should not be corrupted while communicated and should not be changed. Time dead line: The data should be available within a predefined or dynamically allocated period of time. Continuity, Order and Correctness: The data should be available to the consumer continuously, same in the order and the data should be same as before it is communicated. [8]

There are mainly two types of real time systems, Hard Real Time Systems: the completion of an operation after its deadline is considered useless - ultimately, this may cause a critical failure of the complete system. Soft Real Time Systems or Near the Real Time Systems: A soft real-time system on the other hand will tolerate such lateness, and may respond with decreased service quality (e.g., omitting frames while displaying a video).

This research topic mainly covers Soft Real Time Data, e.g. voice data and the protocol that is dealing with soft real time systems, i.e. RTP.

4.1 Real Time Transport Protocol

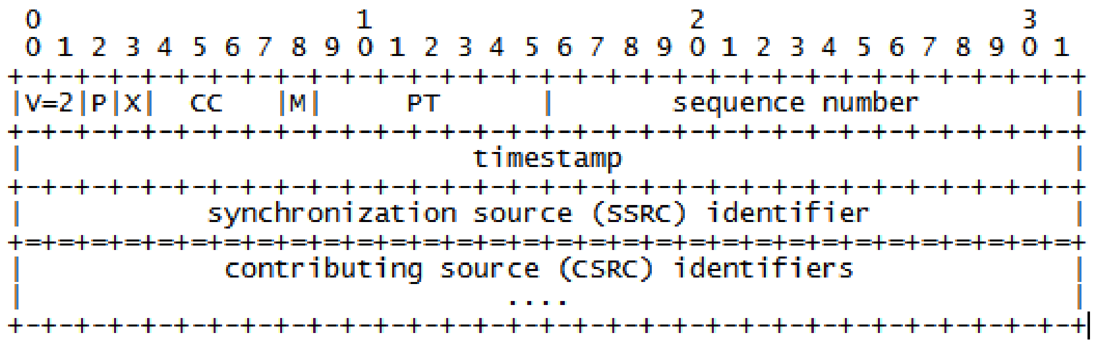
Real Time Transport Protocol is used to transport or communicate soft real time data. RTP is specially designed to communicate audio or video data that is near the real time in nature. RTP provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. RTP ensures that the data receiver should be able to receive and maintain the data with the real time characteristics of data. The services provided by RTP include payload type identification, sequence numbering, time stamping and delivery monitoring. RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services to do so. It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence. The sequence numbers included in RTP allow the receiver to reconstruct the sender's packet sequence, but sequence numbers might also be used to determine the proper location of a packet.

RTP protocol consists of two data link layers i.e. RTP and RTCP, which provide the end to end data delivery and control information as well. The RTP packets carry the data itself (which is of real time characteristics) along with Meta information e.g. Packet sequence number, payload type, timing information etc. RTCP packets carry the control information related to RTP session which is periodically communicated between participants. RTCP main function is to provide 1: Feedback about the delivery of RTP data, 2: Remote end identification to track the changes in Recipient's state, 3: Transmission rate control and 4: Optionally the session control information to control the session. The Receiver of RTP packets sends Receiver Reports and Transmitter sends the Sender reports.

4.1.1 RTP Data Transfer Protocol

RTP Data Transfer Protocol consists of RTP payload which carries the real time data e.g. Audio or Video encoded using a pre-negotiated coded e.g. G729, G723 etc. for audio and H263, H264 etc. for Video.

Figure 3 illustrates and explains the RTP payload header.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: RTP payload header format

As shown in Figure 3, RTP payload header consists of 32 bit chunks containing information about the version, padding, extensions, sequence number, timestamp and identifiers. Table 1 explains the each filed separately.

V: Version, 2 bits	RTP version number which is always set to 2
P: Padding, 1 bit	Represents that if this packet contains any padding bytes at the end which are not part of the payload but some encryption algorithms requires them to make RTP packets of fixed size. The last byte of the padding contains a count of how many bytes are

	padded that should be ignored.
X: Extension, 1 bit.	If set, tells that an extension header follows the fixed header.
CC: CSRC count, 4 bits	The number of CSRC identifiers in the end of fixed header.
M: Marker, 1 bit	The interpretation of the marker is defined by a profile. This is for example used for voice talks to define the talk spurt.
PT: Payload Type, 7 bits	Identifies the format of the RTP payload i.e. the codec used to encode the data contained by this payload.
Sequence Number: 16 bits	Represents the sequence of each RTP packet, which is Initially started from a random number and incrementing by one for each RTP packet. The reason for starting from a random number is to make plain text attacks more difficult in case if encryption is not used.
Timestamp: 32 bits	Resents the time when this RTP sample is taken and computed based on the sampling or framing intervals of transmitter.
SSRC: Synchroniza-tion source, 32 bits	SSRC Is a 23 bit random number value used to label the stream to Identifies the synchronization source. For example in confer-encing system streams from each participant could be identified and handled separately.
CSRC: Contributing source, 32 bits	Identifies the contributing sources in the stream packet which is mixed by mixer from each source. Maximum 15 CSRC contrib-uting source can be identified. The number of identifiers is given by the CC field. CSRC identifiers are inserted by mixers, using the SSRC identifiers of contributing sources. For example, for audio packets the SSRC identifiers of all sources that were mixed together to create a packet, are listed, allowing correct talker indication at the receiver.

Table 1: RTP Header Fields

RTP Payload header is followed by RTP payload data which is the actual real time data (which is normally encoded by some pre negotiated codec e.g. G723, G729 etc.) carried by the payload.

4.1.2 RTP Control Transfer Protocol

RTCP that is RTP Control Transfer Protocol is used to periodically communicate control information related to RTP stream to all participants involved in an RTP session using the same distribution mechanism as the original RTP data packets. The main reason for RTCP is to provide data distribution quality feedback related to flow and congestion control etc., to carry persistent transport level identifier i.e. CNAME and optionally to communicated minimal session control information in a loosely controlled sessions. [9]

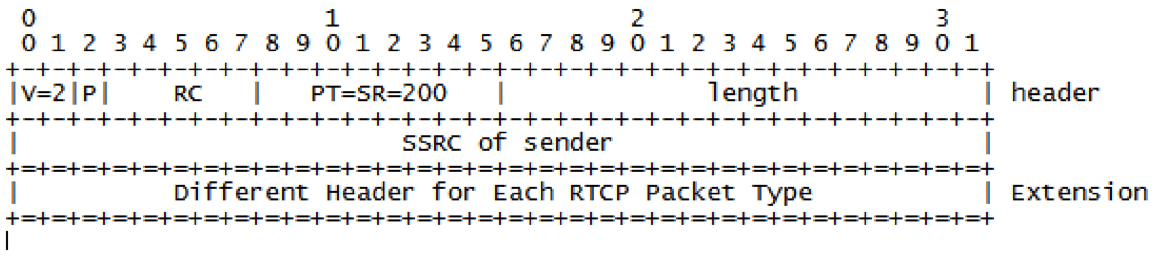
Table 2 shows the different RTCP packets which are communicated in a RTP session

SR: Sender report	Sent from participants which are active senders to communicate the transmission and reception related statistics to the participants that are active receivers
RR: Receiver report	Sent from participants which are receivers and not active senders to communicated the reception statistics to the participants which are active senders
SDES: Source description items	including CNAME
BYE	Indicates end of participation
APP	Application specific functions

Table 2: RTCP Packet Types

Each type of RTCP packet explained above begins with a fixed part which is the same as the RTP payload header and followed by a header extension of variable length specific to the RTCP packet type. [9].

Figure 4 illustrates and explains the RTCP header.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|    RC   |   PT=SR=200   |             length            | header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SSRC of sender                        |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|          Different Header for Each RTCP Packet Type           | Extension
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|
```

Figure 4: RTCP header format

As shown in Figure 4, RTCP header consists of 32 bit chunks containing information about the version, padding, extensions, length and identifiers.

4.2 RTP and Transport Layer Protocols

RTP relies on underlying transport protocol to carry RTP packets called RTP payload. Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services; both protocols contribute parts of the transport protocol functionality. However, IETF recommends that RTP may be used with other suitable underlying network or transport protocols. Therefore, in this research, Web services are chosen as a transport layer for RTP [9].

4.2.1 RTP over UDP

UDP (User Datagram protocol) over IP is the most commonly used underlying network transport protocol which works as an interfacing layer between IP protocol controlling IP networks and application layer protocol like RTP. [10]

UDP operates at the transport layer in OSI and TCP/IP protocol stack and datagram oriented end to end transport by utilizing the power of underlying provides packet switching networks i.e. IP Networks. UDP protocol operates in connectionless mode and provides unreliable transport of data packets known as UDP datagram which means that endpoints do not establish an end-to-end session prior to data transmission and transfer datagram independently on IP network. UDP datagrams are not acknowledged neither retransmitted in a case of error occurred during the transmission of data end to end. Hence UDP protocol provides unreliable, connectionless, not ordered delivery of data packets over IP networks. [11]

The UDP protocol is so far considered to be the best candidate as a transport layer protocol because of the following characteristics:

Lightweight means that UDP does not add too much communication overhead to data communications in a way that it is unacknowledged, connectionless and unreliable protocol. In addition to that, the UDP protocol does not perform any type of congestion control to data communications between two end points, which is beneficial for the applications such as RTP so that one slow connection should not have any impact on a streaming server while remaining connections are performing faster.

Unreliability means that data is sent in the form of packets which is not guaranteed to be delivered to its final destination and recipients do not acknowledge the delivery of packets, which is beneficial in the case of RTP because RTP protocol carries data which is time critical and will lose its effectiveness or usefulness if not delivered in time and error in one particular packet should not hinder the whole communication stream. Unacknowledged delivery of packets helps RTP to establish multicasting or broadcasting based communications of data such as audio or video.

Connectionless means that UDP does not require any session establishment prior to data transmissions which reduce the overhead of handshake related communications and makes the data transfer simpler and faster. RTP takes the advantage of connectionless property of UDP such that applications can transmit its packet without taking into account that there is any recipient which is receiving the transmissions which is very practical in broadcasting type of applications such as IPTV etc.

Multicasting is the technique provided by the IP protocol to allow one-to-many (relationship between endpoints) based communications in the IP based networks. UDP is the most suitable protocol on top of the IP layer because of its connectionless, unacknowledged and in ordered packet switching. Multicasting is an important factor for choosing UDP as transport layer protocol for RTP and applications (e.g. Like IP TV) can broadcast or multicast the real time stream of data by utilizing underlying network’s multicasting feature which works best when UDP protocol is used.

Because of the above explained properties, UDP does not add extra communication overheads which make it lightweight, fast and simple to implement for real-time traffic. In addition, UDP gives the opportunity to a real time application to benefit from the transport layer multicasting and broadcasting features.

4.2.2 RTP over TCP

As described in the beginning of this section, the Real-time Transport Protocol is independent of the transport layer protocol and UDP is the most commonly used transport layer protocol but it is also possible to transport RTP over TCP along with its possible drawbacks. Transporting RTP over TCP involves the communication overhead because of its characteristics listed below:

Reliability means that data should not be damaged, lost, duplicated or delivered out of order. In order to achieve reliability, the TCP layer transport protocol maintains a TCP header attached with each package, which includes a sequence number for each packet, cyclic redundancy checksum etc. The sequence number helps to correctly order the received packages at the receiver end and eliminate duplicated received packages. The CRC checksum is used to identify the damaged packages and negatively acknowledge them resulting into package complete retransmission. Reliability affects the real-time characteristics of RTP data by adding the parameters explained earlier and acknowledging each package or group of packages.

Flow Control is used to control the flow of data between two ends so that the receiver should not be overwhelmed by data transmitter and to prevent the transmitter from sending too many packets to overflow the network resources and the receiver's buffers. In order to achieve this, the TCP implements a sliding window algorithm in which the window size (either set by the congestion window of the network or receiver advised window size) determines the maximum amount of data the sender can send without acknowledgement. The follow control can badly affect the real-time characteristics of data because of delays mainly to the network congestion and end-to-end flow control.

Session maintenance and Multicasting is another drawback of RTP over TCP because TCP protocol is a connection oriented protocol, which means that prior to any data transfer a handshake is needed between the transmitter and receiver in order to establish a connection/session. Each session/connection is uniquely specified by a pair of sockets identifying its two sides. Once the session is established, the transmitter and receiver can exchange data along with the session maintenance traffic e.g. TCP session keep alive etc. Once the data exchange between the two parties is over the session is torn down by another handshake protocol. Due to the connection orientation in TCP protocol, it is not possible to have any multicasting connections and multicasting transmission on multicasting addressing to be received by any number of receivers. However, TCP layer multiplexing overcomes this issue so that a single socket can be simultaneously used for a number of connections/session at a time. Each connection will involve its own connection establishment, flow control and session maintenance. [12]

Despite of all the above mentioned drawbacks of transporting RTP over TCP, there are certain benefits which can only be achieved by using TCP as the underlying transport protocol i.e. networks firewall interoperability, data reliability and transport layer security.

4.2.3 RTP over SOAP/HTTP

As described in Chapter 3, the SOAP framework uses HTTP as the application layer protocol and TCP as the transport layer protocol, therefore transporting RTP over SOAP inherits all the overheads involved in TCP as a transport protocol. In addition, the following overheads are added.

Binary Data to ASCII Encoding is needed to convert the binary RTP payload to textual string in ACII format so that it can be encapsulated in a soap package. For this purpose one can use Hex Encoding which creates the ASCII output of 200% of actual size and also Base64 encoding which produces 150% ASCII output for binary data. Hence if one uses Base64 algorithm then encoding involves at least 50% of actual payload size overhead.

For example:

```
TWFuIGl-
zIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlzIHNp
bmd1bGFyIHBhc3Npb4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2YgdG
hlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGl
5IGNhcm5hbCBwbGVhc3VyZS4=
```

SOAP Header and Body adds a fixed size XML formatting overhead on top of (Base64 or Hex hashed) binary data to represent the RTP payload in soap format. It involves SOAP envelope, SOAP body and necessary RTP header details and RTP Payload.

For example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <SOAP-ENV:Body>
                <ns1:SOARTP xmlns:ns1="urn:SoapRTP">
                        <Payload>
                           TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5v
                            dCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1
                            dCBieSB0aGlzIHNpbmd1bGFyIHBhc3Np
                            b4gZnJvbSBvdGhlciBhbmltYWxzLCB3a
                            GljaCBpcyBhIGx1c3Qgb2YgdGhlIG1pbm
                            QsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2U
                            gb2YgZGVsaWdodCBpbiB0aGUgY29udGl
                            5IGNhcm5hbCBwbGVhc3VyZS4=
                       </Payload >
               </ns1: SOARTP>
        </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

There can extra parameters added to SOAP body which can represent the whole RTP header e.g. Payload type, codec, payload length etc.

HTTP Header and Body add a fixed size overhead on top of XML formatted SOAP message which includes pre-defined header fields i.e. name/value pairs ending with CR/LF. HTTP body is used to carry the SOAP formatted data. Transfer-Encoding field is used to specify the type of content. [13]

Example HTTP SOAP Request:

```
POST /SoapRtp HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <SOAP-ENV:Body>
               <ns1:SOARTP xmlns:ns1="urn:SoapRTP">
                       <Payload>
                          TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5v
                          dCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1
                          dCBieSB0aGlzIHNpbmd1bGFyIHBhc3Np
                          b4gZnJvbSBvdGhlciBhbmltYWxzLCB3a
                          GljaCBpcyBhIGx1c3Qgb2YgdGhlIG1pbm
                          QsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2U
                          gb2YgZGVsaWdodCBpbiB0aGUgY29udGl
                          5IGNhcm5hbCBwbGVhc3VyZS4=
                       </Payload >
               </ns1: SOARTP>
        </SOAP-ENV:Body>
  </SOAP-ENV:Envelope
```

HTTP SOAP Response:

```
POST /SoapRtp HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
```

```
        <SOAP-ENV:Body>
                <ns1:SOARTPResponse xmlns:ns1="urn:SoapRTP">
                        <--0It may contain RTP receiver reports-->


                </ns1:SOARTPResponse >
        </SOAP-ENV:Body>
  </SOAP-ENV:Envelope
```

HTTP Protocol Semantics define how to use SOAP with HTTP as an embedded content within a variety of HTTP requests methods. Basically all HTTP request models or methods work in a request or response message model where HTTP request contains the SOAP request message and naturally HTTP response contains the SOAP response. SOAP Action is used to identify the intent of HTTP request contains SOAP message that is represented in the form of a URL.

```
soapaction    = "SOAPAction" ":" [ <"> URI-reference <"> ]
URI-reference = <as defined in RFC 2396 [4]>
```

HTTP Post Request is typically used to transport SOAP messages to the SOAP service where SOAP service executes the request and sends the response embedded in HTTP response which is 200OK in case of success and In case of failure the HTTP error codes are used. SOAP Fault object can also be sent if the Error is beyond the HTTP transport layer and lies within SOAP framework boundaries. [14]

Example SOAP Using HTTP POST request/response Method:

```
POST /StockQuote HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://electrocommerce.org/abc#MyMessage"

<SOAP-ENV:Envelope...


HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope...
```

It is also possible to use HTTP Extension Framework in order to identify the presence and intent of a SOAP HTTP request. Whether to use the Extension Framework or plain HTTP typically depends upon the policy and capability of the communicating parties. Clients can force to use of the HTTP Extension Framework as SOAP transportation method by using a mandatory extension declaration and the "M-" HTTP method name prefix.

Example SOAP Using HTTP POST request/response Method:

```
M-POST /StockQuote HTTP/1.1
Man: "http://schemas.xmlsoap.org/soap/envelope/"; ns=NNNN
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
NNNN-SOAPAction: "http://electrocommerce.org/abc#MyMessage"

<SOAP-ENV:Envelope...

HTTP/1.1 200 OK
Ext:
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope...
```

Servers can force the use of the HTTP Extension Framework by using the 510 "Not Extended" HTTP status code. That is, by using one extra request and response round trip (i.e. forth and back); either party can detect the policy of the other party and act accordingly. [14]

This chapter elaborated the Real-Time data communications by exploring the nature of such type of data, the protocols used for its communication and most importantly the transport protocols used to carry such kind of data. The following chapter focuses on the architectural details concerned with transporting real time data on the Web Services platform by providing the implementations level details. The design techniques and principles provided in the following chapter could be used as a guideline for designing, implementing and deploying the web services capable of communicating real-time data.

5 ARCHITECTURE AND DESIGN PRINCIPLES

The protocols carrying real time data, such as RTP, offer real time data delivery in different ways e.g. send only, receive only or both send receive in full duplex mode. Secondly, the main characteristic, unlike in the traditional web service communication model is the continuous flow of data within one particular session. In addition, Real Time transport protocols require the control signaling, e.g. RTCP packets, to be exchanged periodically within the whole real time communication session lifetime. This chapter explores the Web Services architecture to address the real time data traffic challenges and proposes a solution within the web services context to those challenges.

5.1 Traditional Web Services Communications Model

The basic Web Service communications architecture defines the ways the software agents in a Web Service Model interact with each other to exchange data. Following are the three different roles which can be taken in basic web service architecture.

- Service Requester: The client component which requests XML/SOAP content offered by web service

- Service Provider: The server component or the process which processes the XML/SOAP requests and generates the XML/SOAP response back to service requester

- Discovery agency: Agency through which a Web service description is published and made discoverable

Figure 5 illustrates the Service Oriented Architecture.

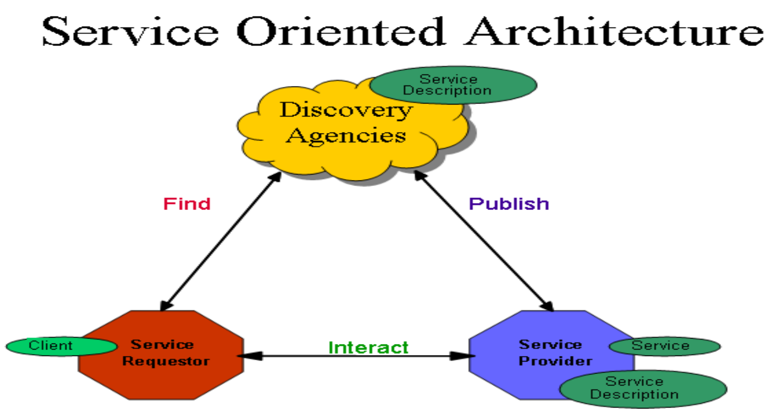


Figure 5: Web Service architecture

In this basic architecture, the service requester role and service provider role both can be played by the same software component at a time. In order to explain it further, following are two of the interactions possible between two software components using web services or service oriented architecture to exchange data with each other. [15]

5.1.1 Client Server Model

The client server model is the most commonly used model in basic service oriented architecture where one software component acts as Service Provider and have access to database and business logic components to process and fulfill request sent by Service Request which is a client software component.

Figure 6 illustrates and clarifies the Client Server Communication Model.

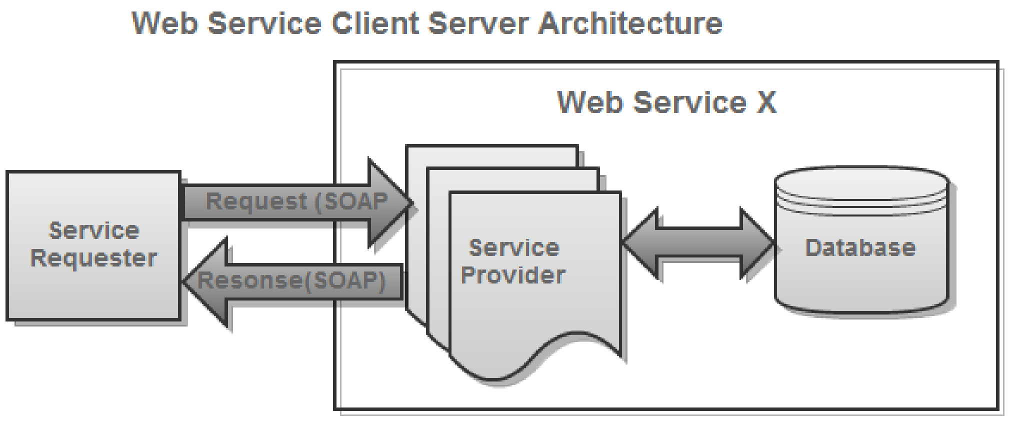


Figure 6: Web Service Client Server Communication Model

As shown in Figure 6, the Service Requester sends a SOAP based request to the Service Provider running a HTTP based service that is served by retrieving data from data base and a SOAP based response is sent back to the Service Requester.

5.1.2 Peer to Peer Model

In this communication model, both software components have a dual role, i.e. Service Provider as well as Service Requester for the other opponent. This type of model is usually implemented between services to exchange data on a continuous basis and both sides implement or host a service to produce data for the other opponent.

In this model the request and response are handled separately for both services and a different communication channel of the underlying transport protocol is used for each direction, i.e. TCP port in the case of HTTP. In the case of HTTP the requests and responses are not correlated and it is the responsibility of the application layer to correlate the correct request with the correct response.

Figure 7 illustrates the Peer to Peer Communication Model.

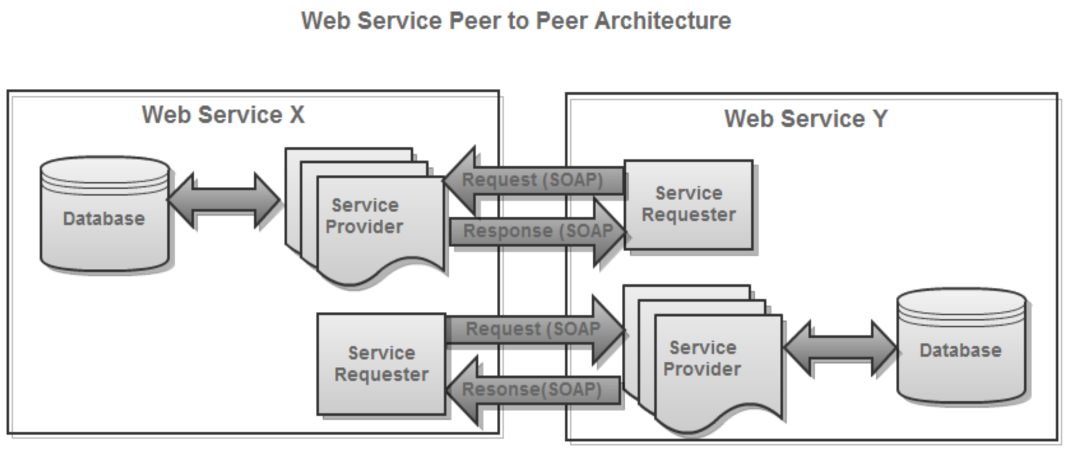


Figure 7: Web Service Peer to Peer Communication Model

Figure 7 shows that in a Peer to Peer Communication Model the communication between the Service Requester and Service Provider is the same as in Client Server Architecture but in this case both ends play the roles of both Service Provider and Service Requester.

The Web Service discovery mechanism is out of the scope of this discussion and is not be covered here because there is a separate protocol to accomplish it and that is independent of the data carried by the Web Service.

5.2 Real-time Web Services Communications Model

In order to adopt Service Oriented Architecture as the main communication means for real time data e.g. Voice and Video, there are several approaches to address the communications challenges imposed by the data because of its real time nature. In addition to that, there are several features that come along with real time data and become necessary to use when dealing with this type of data communications, e.g. signaling, communication modes etc. In the following subsections those real time data challenges are discussed and different approaches are offered to address the issues.

5.2.1 Real Time Session Establishment

Real-time data transport protocols e.g. RTP require the session to be established before the actual data exchange in which there are several parameters that need to be negotiated. E.g. codec that will be used to encode/decode data, data direction i.e. weather it is send only, receive only or bidirectional etc. For that purpose there are several industry standards signaling protocols being used e.g. SIP, H323 etc. In addition to these protocols separate customized SOAP based customized signaling can be implemented to subscribe a session to the other opponent and the same is done from that side.

5.2.2 Real-time Web Services Communications Techniques

Real-time data e.g. voice or video in its main characteristics is a continuous stream of data for a pre-defined duration or an event based duration. In order to use web services platform as transportation means for the continuous flow of data there are the two following possible ways to achieve data continuity.

- Continuous Request and Response

  When two parties (i.e. A Party and B Party) want to establish a real time data communication session and start streaming to each other then they exchange Web Service URLs where streams will exchange with each other. Once the streaming session begins, the A Party sends a HTTP Post Request with SOAP message containing RTP data in its content and B party's web service processes the request by consuming the RTP data and sends 2XX response in case if there is no error occurred while consuming the data, See Fig. 8. It may respond to any error code if B party for some reason is not able to accept the data e.g. if the codec that is used to encode data is not supported. A Party keeps on sending HTTP Post request periodically depending on packet size, data rate, frame rates etc. The rate on which each new request is sent will have a great impact on the real time effectiveness of the data.

Figure 8 illustrates and exemplifies the Continuous Request and Response Model.

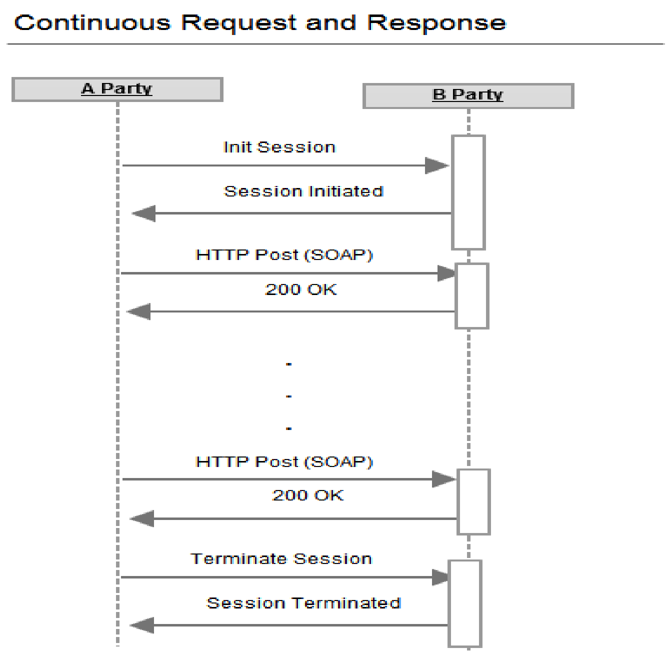


Figure 8: Real- time Web Service Communication Model

As Figure 8 shows, this model involves three phases, the session initialization phase to setup the session, the continuous request and response phase that repeats until the lifetime of the session and the session termination phase to end the communications.

As discussed in the earlier section, this method requires the RTP over SOAP session to be established first using the SOAP based customized signaling protocol or some industry standard, for example SIP or H.323.

- Request and Continuous Response

In this mode, two parties involved, A Party and B Party, adopt two different roles of traditional service oriented architecture, i.e. service provider and service requester. The A Party which is the service requester in this case knows the web service URL of the B Party which is the service provider. A Party sends a HTTP GET request to the B Party's web service and in B party processes the request and sends a 206 PARTIAL DATA response in a periodic fashion. The body of HTTP 206 response contains the SOAP message containing RTP data in its contents and A Party receives the 206 response and consumes the RTP data. HTTP 206 PARTIAL DATA response is sent until there is RTP session and when closing down the session the last response is 200OK which means that there will not be any more data coming for this particular RTP over SOAP session.

Figure 9 illustrates the Request and Continuous Response Model.

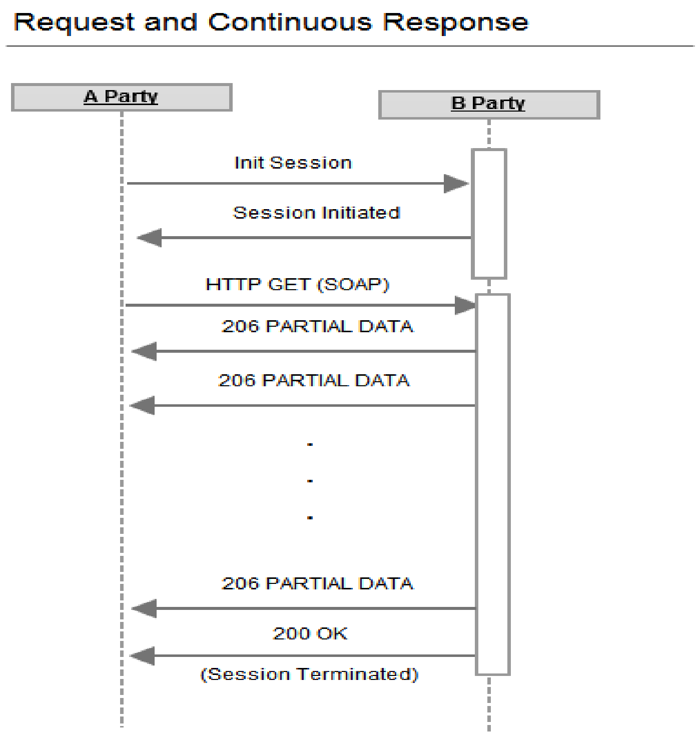


Figure 9: Real- time Web Service Communication Model

Figure 9 shows that this model involves three phases, the session initialization phase to setup the session, the request and continuous response phase that repeats until the life-time of the session and the session termination phase to end the communications.

5.2.3 *Stream Directions*

Stream direction is one of the most important characteristic that defines the data sender and receiver roles in one particular session. There are typically three different combinations in a typical two party conversation i.e. Send Receive, Send Only or Receive Only. Whereas Send Receive configuration works for Full Duplex channels and other two combinations are for Half Duplex channels.

- Half Duplex or Unidirectional Streams

  The streams in which only one party is the sender and the other is the receiver are called Half Duplex or Unidirectional Streams. This kind of communication model fits well into a web service context so that the data sender party takes the Service Provider role in the web services context and the data receiver party takes the Service Requester role. Half duplex steams can use both the Continuous Request and Response Communication Model and Request and Continuous Response Model of Web Services. Typical use of Half Duplex streams is for recording applications, play back application e.g. radio, music etc., voicemail applications etc. which need both recording and playback but not at the same time.

- Full Duplex or Bidirectional Streams

  These types of streams need the active data exchange from both parties, which means that both parties A Party and B Party will be sending and receiving data at the same time. The most suitable real time web service communication model for full duplex streams is the Continuous Request and Response Model where each party subscribes to the other party's stream and receives a continuous stream of SOAP requests and responds them. The other real time web service communication model can also work for Full Duplex Streams but this model will require a separate communication channel to deliver reports which are discussed in detail in the following section. Full Duplex or Bidirectional Streams are mostly used for applications e.g. voice, video or text chat clients where both parties are actively exchanging real time data.

As discussed in the earlier section, Stream directions need to be negotiated between the parties beforehand and can be chosen in any combination e.g. Send Receive, Send Only or Receive Only. The Communication Model that is used for such communication may not be necessarily possible to be chosen on run time and should also be negotiated with other parameters.

5.2.4 Sender and Recover Reporting Mechanism

Real-time Transport Protocols such as RTP along with regular data transfer requires control information i.e. RTCP packets exchanged between the involved parties periodically and contains the information about Data Flow Control and other statistics required to control the real time data. RTCP in particular requires the participants to exchange receiver reports only if both parties are active senders and if one participant is sender only then that party is required to send Sender Reports and other party will send Receiver Reports. Mainly there are two different approaches to implement control packet flow e.g. RTCP packets exchanges which are explained below:

- Explicit Control Information Exchange

  One approach to implement Real Time Transport Control Protocol i.e. RTCP for a web service based platform is that RTCP information is to be explicitly exchanged along with RTP data itself. Explicit information exchange can be done within the same communication channel with a different SOAP message or out of channel like the native RTP implement does by using a different communication port. In channel RTCP reporting is only possible for Full Duplex Streams because it does not require the sender to send Sender Reports. This approach is applicable to both communication models for real time web services and the one and only option for the Request and Continuous Response Model.

- Implicit or Embedded Control Information Exchange

  The other way to implement RTCP functions in a web service based platform is to utilize the HTTP response mechanism that is the HTTP 200 OK response containing the RTCP control information. This method can be only used for two way communications i.e. for Full Duplex Streams because such type of streams involve only receiver reports that can be embedded in the HTTP 200 OK Response.

The reporting mechanism of the Real time transport control protocol should not be adopted on run time because different approaches are not compatible to all kinds of communication models and communications modes.

5.2.5 Multicasting and Broadcasting

Multicasting and Broadcasting is an important feature of Real Time Transport Protocol which is accomplished in some cases by underlying transport protocols or by an application layer (i.e. Transport Protocol itself or a Signaling Protocol). Due to the fact that the Web Services platform is based on the SOAP protocol (using TCP as transport layer protocol without broadcasting support), broadcasting is not possible in Real Time Web Services and the application will have to rely on application layer protocols to accomplish multicasting.

- Multicasting with Half Duplex or Simplex Streams

Multicasting in this context means that there are more than one recipients of a real time stream which is in offer by a web service provider. This can be achieved by using both real time web service communication models.

While using the Request and Continuous Response Model, each recipient playing a service requester role will request the stream by using the HTTP GET Method and stream service provider i.e. Streaming Server will use the continuous response communication model for sending SOAP encapsulated RTP packets. [13]

Whereas when using the Continuous Request and Response Model, there will be one Streaming Server which will keep track of currently active stream clients and will act as the Service Requester and the Streaming Clients will play the Service Provider role and will receive the RTP data in SOAP request. In order to join the stream each Streaming Client will have to subscribe for a session with the Streaming Server and will then be able to receive RTP SOAP requests.

Figure 10 illustrates how the Multicasting is implemented.

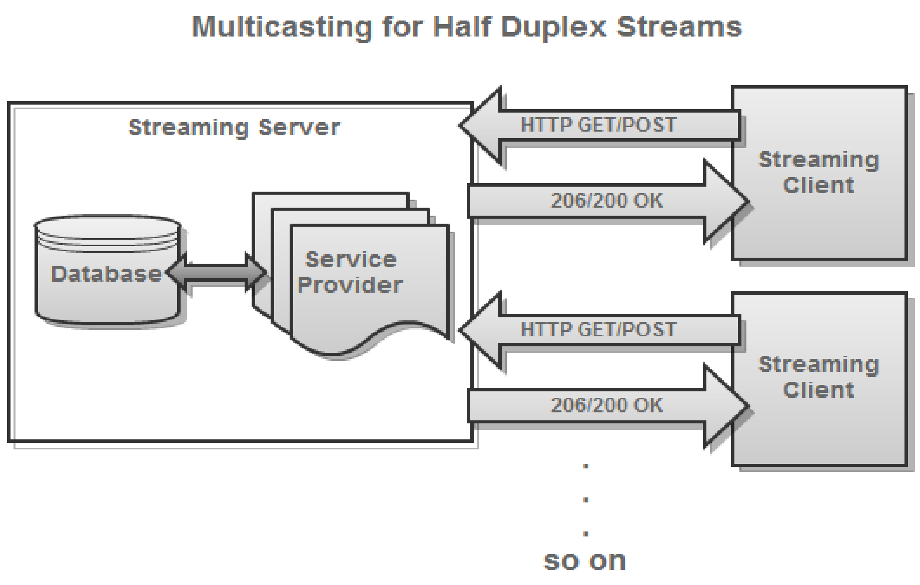


Figure 10: Real- time Web Service Communication Model

As shown in Figure 10, each participant in the multicasting session establishes a session with a relaying server based on either Continuous Request and Response Model or Request and Continuous Response Model.

- Multicasting with Full Duplex Streams

  Because in Full Duplex Steams all participants are active data transmitters in this case the data need to be mixed together to allow more than two participants engaged in the same session. In order to implement data mixing there is a need of a dedicated data mixer acting as a Media Server that will also be based on Real Time implementing Web Service Communications Architecture to communicate the real time data. All the streaming clients will send their Real Time Data to Media Server and Media Server in response will send it back mixed RTP content received from other participants.

Figure 11 illustrates how the Multicasting is implemented for Full Duplex systems.

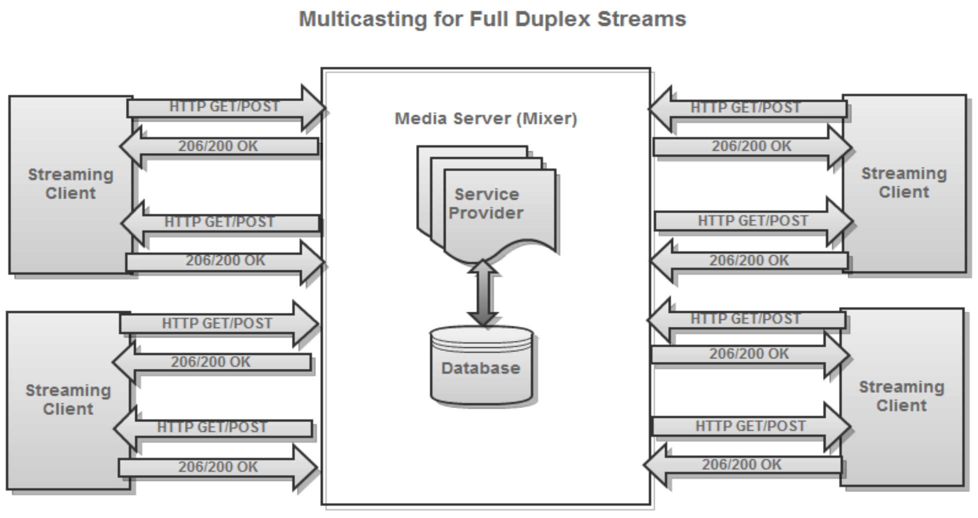


Figure 11: Real- time Web Service Communication Model

As shown in Figure 11, each participant in the multicasting session establishes a full duplex session with a relaying server based on either the Continuous Request and Response Model or the Request and Continuous Response Model.

5.3 Comparison Matrix between Models and Techniques

In the previous sections there are different aspects for real time data communications discussed and each of the issues is addressed with different approaches or solutions in the context of the Web Services Platform. Following is the summary and comparison matrix where different approaches are listed.

Participants	Communication Direction	SOAP/HTTP Transportation Mode	Reporting Methodology
Two Parties	Half duplex or One way Communications	Continuous Request and Response Mode	Explicit Control Information Exchange
			Implicit or Embedded Control Information Exchange
		Request and Continuous Response Mode	Explicit Control Information Exchange
	Full duplex or One way Communications	Continuous Request and Response Mode	Explicit Control Information Exchange
			Implicit or Embedded Control Information Exchange
		Request and Continuous Response Mode	Explicit Control Information Exchange
Multi party or Multicasting	Half duplex or One way Communications	Continuous Request and Response Mode	Explicit Control Information Exchange
			Implicit or Embedded Control Information Exchange
		Request and Continuous Response Mode	Explicit Control Information Exchange
	Full duplex or One way Communications	Continuous Request and Response Mode	Explicit Control Information Exchange
			Implicit or Embedded Control Information Exchange
		Request and Continuous Response Mode	Explicit Control Information Exchange

In this matrix, the columns represent the communications characteristics i.e. participants, communication direction, communication mode and reporting methodology whereas the rows represent the possible options for the characteristics. Each row entry in the right side column represents the possibility for the entry in left column at same row.

5.4 Web Service Descriptions

This section describes the way how the RTP data could be formatted in the XML format based on SOAP standards. The examples given in this section gives a guideline about how to transform RTP payload fields into SOAP standards and can be adjusted based the communication model being used.

RTP Data packet can be divided into two parts, which is the header containing all the RTP header information and the payload which is containing the actual RTP data. The SOAP method used for data packet is “SOARTPDataPacket” that indicates the receiving end that it is a data packet and can be treated accordingly.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <SOAP-ENV:Body>
                <ns1:SOARTPDataPacket xmlns:ns1="urn:SoapRTP">
                        <Header>
                                <Version></Version>
                                <SequenceNumber></SequenceNumber>
                                <SSRCIdentifier></SSRCIdentifier>
                                <CSRCidentifiers>
                                        <CSRCIdentifier></CSRCIdentifier>
                                </CSRCidentifiers>
                                <Extensions>
                                        <Extension>
                                                <ID></ID>
                                                <Header></Header>
                                        </Extension>
                                </Extensions>
                        </Header>
                        <Payload>
                           TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5v
                            dCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1
                            dCBieSB0aGlzIHNpbmd1bGFyIHBhc3Np
                            b4gZnJvbSBvdGhlciBhbmltYWxzLCB3a
                            GljaCBpcyBhIGx1c3Qgb2YgdGhlIG1pbm
                            QsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2U
                            gb2YgZGVsaWdodCBpbiB0aGUgY29udGl
                            5IGNhcm5hbCBwbGVhc3VyZS4=
                        </Payload >
```

```
               </ns1:SOARTPDataPacket>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

RTCP Sender Reports are merely based on the header and do not contain any body. The header consists of Version, SenderInfo, ReportBlocks and Extensions elements. The SOAP method used for the data packet is “SOARTCPSR” that indicates the receiving end that it is a Sender Report and can be treated accordingly.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <SOAP-ENV:Body>
               <ns1:SOARTCPSR xmlns:ns1="urn:SoapRTP">
                       <Header>
                               <Version></Version>
                               <SenderInfo>
                                       <SenderSSRC></SenderSSRC>
                                       <NTPTimeStamp></NTPTimeStamp>
                                       <RTPTimeStamp></RTPTimeStamp>
                                       <PacketCount></PacketCount>
                                       <OctetCount></OctetCount>
                               </SenderInfo>
                               <ReportBlocks>
                                       <ReportBlock>
                                               <SSRC></SSRC>
                                               <FL></FL>
                                               <CNPL></CNPL>
                                               <EHSNR></EHSNR>
                                               <IAJ></IAJ>
                                               <LSR></LSR>
                                               <DLSR></DLSR>
                                       </ReportBlock>
                               </ReportBlocks>
                               <Extensions>
                                       <Extension>
                                               <ID></ID>
                                               <Header></Header>
                                       </Extension>
                               </Extensions>
                       </Header>
               </ns1:SOARTCPSR>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

RTCP Receiver Reports, same as RTCP Sender Reports, are also merely based on the header and do not contain any body. The header is the same as the RTCP Sender Report except SenderInfo consists of fewer details (which can be made optional in SOAP format).

The SOAP method used for the data packet is “SOARTCPRR” that indicates the receiving end that it is a Receiver Report and can be treated accordingly.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <SOAP-ENV:Body>
                <ns1:SOARTCPRR xmlns:ns1="urn:SoapRTP">
                        <Header>
                                <Version></Version>
                                <SenderInfo>
                                        <SenderSSRC></SenderSSRC>
                                </SenderInfo>
                                <ReportBlocks>
                                        <ReportBlock>
                                                <SSRC></SSRC>
                                                <FL></FL>
                                                <CNPL></CNPL>
                                                <EHSNR></EHSNR>
                                                <IAJ></IAJ>
                                                <LSR></LSR>
                                                <DLSR></DLSR>
                                        </ReportBlock>
                                </ReportBlocks>
                                <Extensions>
                                        <Extension>
                                                <ID></ID>
                                                <Header></Header>
                                        </Extension>
                                </Extensions>
                        </Header>
                </ns1:SOARTCPRR>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

These reports can be embedded into both SOAP request and SOAP response depending upon the communication model being used.

6 EXPERIMENTATION

This section provides implementation details of the prototype implementation representing a real world application of the Web Service Platform as Real Time Transport Protocols. The experimentation involved the whole HTTP stack implementations on top of Windows sockets offering both the client and server side functionality, SOAP protocol implementations, XML parser implementations, Base64 algorithm implementations, Wrapper layer implementation on top of an open source G.729 library and voice capturing and playback module implementations using Windows Multimedia Application Programming Interface WINMMAPI. The programming language used for the prototype implementation was C++ in the Windows platform and the tool used was Microsoft Visual Studio 2005. In addition, the experimentation involved one of the proposed communication models and analysis of its performance and efficiency based on different network parameters e.g. frame rate and packet size etc. The experimentation results are explicitly explained in Chapter 7. Following are the details about the experimental work done.

The experiment based on voice data involved a telephony application transporting voice data over web service based network by capturing voice data from the sound device of a computer and then playing back voice data to the receiving end after decoding it. The communications is based on the communication model presented in Chapter 5. Table 3 shows the different parameters for the voice data used for the experimentation.

Index	Audio Parameter Name	Value
1	Codec Name	G.729
2	Bit Rate	128 kbps
3	Samples per second (Frame Rate)	8.0 kHz
4	Compression Rate	16:1
5	Modulation	Pulse Code Modulation

Table 3: Audio parameters for experimentation

The application used Windows MMC SDK (Windows Multimedia Control Software Development Kit) to capture voice data in WAV format from the sound device and the same for playback, too. The data was further compressed using G.729 codec library.

Figure 12 illustrates the steps involved in communicating voice data over the web service platform.

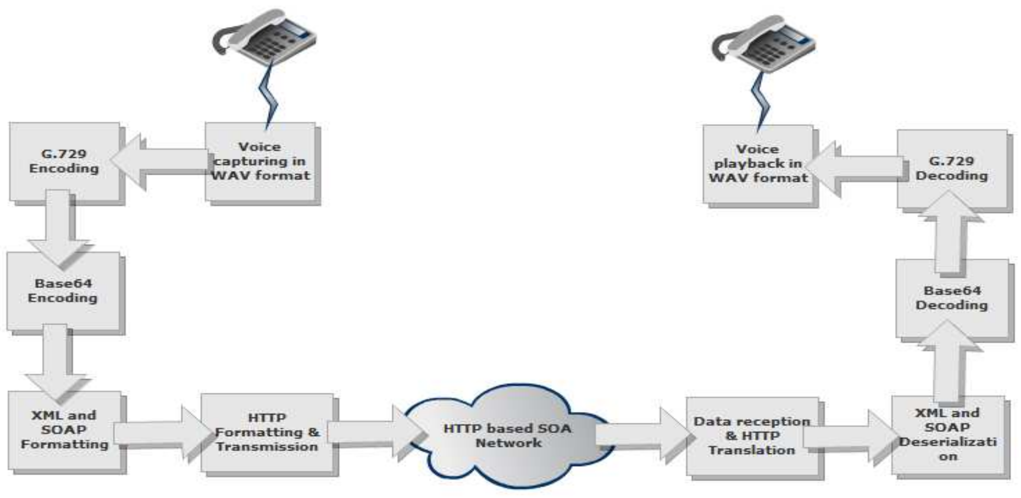

Voice capturing in WAV format
G.729 Encoding
Base64 Encoding
XML and SOAP Formatting
HTTP Formatting & Transmission
HTTP based SOA Network
Data reception & HTTP Translation
XML and SOAP Deserializati on
Base64 Decoding
G.729 Decoding
Voice playback in WAV format

Figure 12: Real- time Web Service based voice communications

The data capturing, encoding, formatting and transporting steps are shown in Figure 12. On the transmitter end after capturing, the intermediate steps involve encoding and formatting it to the SOAP format and transporting it over the web service platform using HTTP as the transport layer protocol. On the receiver end, after receiving it back from the network, it involves de-serializing from the SOAP format, decoding it from G.729 format into WAV format and then playing back using Windows MMC SDK APIs.

Voice Encoding and Decoding

The voice is captured in the WAV format from the sound device of the client workstation using operating system multimedia services and then compressed using G.729 codec. The compressed binary data is then converted to textual data by using base64 encoding that can be used in a SOAP based communication model. The same codecs and multimedia services are used to decode data on the receiving end.

SOAP and XML Serialization and Deserializations:

The textual converted binary data is serialized into XML format in the form of SOAP objects and XML data is formatted which is compatible with the SOAP service and the underlying networks. On the receiving end, SOAP based XML contents are de-serialized and base64 based textual data is retrieved which can be further decoded as explained in Chapter 6.2.

HTTP Transportation:

An HTTP Service is needed to be running on both client workstations that will receive data from both ends and similarly both ends implement the HTTP Client to send or post data towards the opposite end. The HTTP layer also extends and removes the HTTP overheads when transmitting the encapsulated SOAP data on both the transmitting and receiving ends.

There are different types of real time data that could be experimented over the communication model represented in Chapter 5, e.g. voice, audio, video, text, image sharing and content sharing. For this particular experimentation, voice data was chosen because of its importance in multimedia systems. E.g. banks using a 3rd party recording system to record the conversation between their customers and their legal department would need a playback for reference, in case there were a quarrel between the customer and the bank.

7 RESULTS AND ANALYSIS

This section describes the experimental results generated from a software implementation based on the architectural model represented in Chapter 5. The main questions involved in real-time communications, i.e. RTP transported over Service Oriented Architecture, are which kind of communications overhead is involved due to pushing the data communications to a TCP based application layer protocol such as HTTP. In the presence of certain degree of tolerance on communication overhead, how the real time effectiveness is affected. In addition to that, defining the use cases where real-time effectiveness is less important than data reliability.

7.1 Communications Overhead

The Real Time Transports Protocols traditionally uses UDP protocol as the transport layer protocol just because it involves the least communications overhead and makes the real time transmissions efficient and fast. It does not care about features such as reliability or confirm delivery, flow control, error control, ordered delivery etc. which are the main source of communications overhead and mostly offered by connection oriented protocols such as TCP.

7.1.1 Communication Overhead Types

If the RTP is supposed to be communicated over service oriented architecture i.e. web services platform then there are the following three types of overheads involved.

- Fixed Communications Overhead

  Fixed communications overhead involves TCP connection establishment, TCP error and flow control which adds extra bytes to the packet size by TCP layer header, HTTP header and request/response model, SOAP header and body details in the form of XML. The fixed part of the overhead naturally decreases in percentage if the size of real time data chunk (i.e. packet/frame size) is bigger and that makes the frame rate slower. This practically means that applications will buffer data for long time and will send big chunks of data together.

  Fixed Overhead = 12 Extra bytes of TCP Header + Size of HTTP Header + Size of SOAP header and body details + full size of SOAP Response

- Variable Communications Overhead

In addition to fixed size of communication overhead, conversion of binary RTP data to string based data that makes it possible to be used within textual protocol such as SOAP/XML adds 50% (of the actual real time data) overhead. The effect of variable part of overhead is constant in relation to size of real time data chunk i.e. packet/frame size and frame rate.

Variable Overhead = 50% of Real Time Data Packet Size

- Occasional Communications Overhead

Occasional overhead may be added because of occasional network conditions which could cause TCP packet retries and delay the data delivery. One time retry will cost 100% overhead on top of data containing both fixed and variable size overhead. But this overhead will be minimum and approaches zero if the network is fast and reliable such as today's fixed and wireless networks such as 3G or 4G.

One retry = 100% of Real Time Data along with Fixed and Variable Overhead

In summary, the total communication overhead involved in transporting data in real time nature consists of three different components of overheads, fixed overhead involved due to additional formatting, variable overhead based on size of data due to binary to text conversion and occasional overhead which may be involved due to error recovery retransmissions.

7.1.2 Overhead Variations

Communications overhead is very much affected by the variations in Real Time Data Packet Size and Frame Rate. Packet size and frame rate are inversely proportional to each other so that if the frame rate is increased, the packet size is decreased because less data is buffered and it is transmitted immediately. On the other hand, if the packet size is increased it will cause data to be buffered for a longer period of time and transmitted in the form of a bigger chunk.

For real time communications over web service platform, communications overhead is sharply increased by the increase of the frame rate because of a fixed part of communications overhead. By increasing the frame rate, the data packet size is reduced because the data is collected from the data source more often than a fixed bitrate resulting into overhead increased because of fixed size SOAP and HTTP headers will be applied on each small chunk of data and also delivery of each chunk will be acknowledge individually in the form of a complete SOAP response. On the other hand, if the frame rate is decreased and the data is collected for a longer period of time, transmitted in bigger chunks in the form of SOAP message and acknowledged less often causing reduction in SOAP traffic. Low frame rate will significantly reduce the fixed part of communications overhead but if the packet size is too big it may start increasing the overall communication overhead because of the occasional part of the overhead. This is due to a greater risk of errors in transmission and retries especially on error prone networks such as 3G, 2G.

Figure 13 illustrates the overall communication overhead variations.

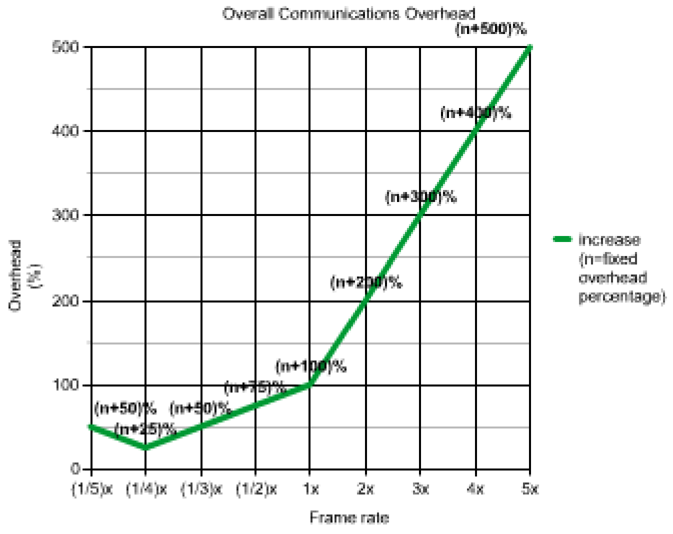


Figure 13: Overall communication overhead variations in the experiment

The graph shown in Figure 13 explains how the overall overhead varies based on the frame rate used for real time data transmissions. The X axis shows the frame rate variations and Y axis shows the overhead variations. It is visible from the graph that increase in the frame rate increases the communication overhead because of the fixed communication overhead. On the other hand, decrease in the frame rate decreases the communication overhead but only up to some extent and then it starts increasing again because of the occasional communication overhead.

7.2 Communication Delays

In comparison to UDP which is a connectionless and unreliable transport layer protocol, the web service platform involves its complexity and overheads which result in the following types of communication delays affecting the real time effectiveness of data.

Computational Delay is involved because in order to transport data on the web services platform, binary real time data need to be encoded into textual form and then encoded into XML format before it is transported over the HTTP application layer protocol. These encodings of data need computation, which may cause some delay on data production and consumptions on both ends.

Transportation Delay is involved due to the fact that every HTTP package needs to be acknowledged with a response in the transport layer TCP. RTP over web services will face a significant level of transportation delay which might have an impact on the effectiveness of data.

7.3 Real-time Effectiveness

Real time effectiveness of RTP data is also very much dependent on Packet Size and Frame Rate. It is directly proportional to frame rate up to some extent i.e. greater frame rate will provide greater real-time effectiveness because the data will be buffered for a shorter period of time and it will be delivered immediately which is beneficial for time critical data. On the other hand, a very high frame rate may actually lower the real-time effectiveness as shown in Figure 14 due to the fact that communication overhead increased sharply by increasing frame rate.

Figure 14 illustrates the real-time effectiveness variations.

.

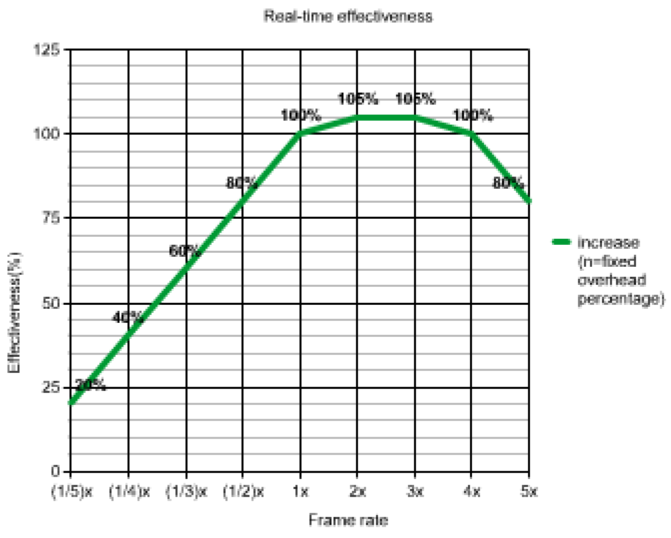


Figure 14: Real-time effectiveness variations

The graph shown in Figure 14 explains how the real-time effectiveness of the data is affected by the frame rate used for real time data transmissions. The X axis shows the frame rate variations and Y axis shows the effectiveness variations. It is visible from the graph that decrease in frame rate decreases the effectiveness because of the communication delays. On the other hand, decrease in the frame rate increases real-time effectiveness but only up to some extent and then it start decreasing again because of the computational delays and communication overheads.

So there should be some tradeoff between the frame rate and real time effectiveness for the optimum result which is very much dependent on the bandwidth and data rate of underlying networks.

7.4 Security and Data Privacy

Security is always one of the biggest concerns when talking about real time data communications because data travels through different kinds of networks and there is a need for data encryption and security so that real time data is not vulnerable to security risks. Traditional RTP communications give the possibility of SRTP (Secure Real-time Transport Protocol) which can be used to secure the data. Using the Web Services platform as the transport layer for real time data communications brings a high degree of security with the help of its TCP layer secure layer, i.e. SSL and TLS.

7.5 Service and Network Interoperability

RTP communications over its traditional UDP transport layer faces greater challenges of interoperability between different networks, passing the data channel through firewall and network address translation issues. In order to overcome such issues there are multiple solution protocols e.g. STUN and UPNP. Web Service platform being a transport layer of RTP data is independent of such shortcomings and there is no need to worry about e.g. network address translation and firewalls, because HTTP is the most accepted and familiar protocol for the firewalls and networks. As discussed in Chapter 5, there are several approaches which can help with such type of challenges.

7.6 Applicability and Benefits

The Web Service platform could be more suitable for the type of applications where data security, reliability and accuracy for real time data are much more important than its real time effectiveness or the situations when real time effectiveness becomes the least important.

Table 4 explains the service comparison by listing the features or use case scenarios that are the handled efficiently by Real-Time communications over Web Services Platform and also at the same time drawbacks involved for traditional UDP based Real-Time communications.

Feature	Benefit of Web Service Platform	Drawbacks from Traditional UDP based Communications
Unified Data Communications	Real time communications over Web Service platform will inherit all the benefits of web standardizations and unifications which are generally most acceptable means for integration	Applications have to deal separately for RTP streams and have to face the complexity of dual interfaces to deal ordinary data and real time data.
Interoperability	HTTP brings the best interoperability and there will be the same channel for ordinary and real time communications.	There is a need for separate firewall management for real time data and need for additional services e.g. STUN, UPNP etc. to deal with this.
Reliable and Secure Communications	TCP brings the lossless communications of data which is real time as well as business critical at the same time	RTP over UDP doesn't provide the data reliability and difficult network conditions may cause data loss which might not be not acceptable for some businesses.

Table 4: Service comparison

Real life examples for such cases where web services used as a real time communication platform could be e.g. Business Voice Recording applications for contact centers where loss of data due to network conditions could be harmful for the business, Voice Mail recording and playback applications etc.

8 DISCUSSION AND CONCLUSIONS

This section focuses on direct answers to the research question, i.e. how feasible is the proposed communication model based on Web Service Platform for real time data communications in general and multimedia applications in particular.

The Information and Communication Technology has evolved over the period of time in all directions including processing and computing power, memory management and data communications and today's Information and Communication Technology world is moving in the direction of automation, convergence and integrations and the Service Oriented Architecture is the most acceptable industry standard based on the Web Services Architecture. On the other hand, traditional protocols for transporting and communicating real time data e.g. multimedia communications for audio, video, or other type of media, are not able to fully benefit from the latest developments in ICT and also bring challenges in the SOA context when the services need to deal with this type of data. This research provides a guideline on how to transfer those types of real time communications also on the modern SOA platform so that the interoperability of the services would become standardized and unified.

Web is becoming the standardized platform for businesses and services to interoperate and so is the web based technologies e.g. HTML5, XML, Web Services and SOAP. From the current trends, it is clear that there is a need of capability of such technologies to carry real time data to integrate multimedia and communications applications.

Despite the fact that the Web Services platform or technology brings its own communication overhead and data transfer delays, considering the bandwidth and data transfer speed of today's communications networks along with computational power that modern computers possess, the communications overheads are negligible. E.g. in the early days with Voice over IP (VoIP) technology that is one of the most famous use cases for real time transport protocol, the data networks were slow and UDP was chosen due to its least communications overhead and data continuity was more important than the reliability. In today's world bandwidth and data transfer speeds have been improved significantly so that the impact of communications overheads is not significant.

The experimentation described in Chapter 6 showed that data transfer over the web services platform for voice data captured at 128kbps bit rate would add a variable communication overhead of 50% for base64 encoding and some fixed communication overhead of

SOAP, XML and HTTP header. It could make it roughly 4 times bigger than the original data, i.e. 512 kbps which is very reasonable for the available networks today.

Service Oriented Architecture (SOA) or Web Services based on SOAP/XML has become a worldwide accepted standard for data sharing among business services, e.g. accounting, finance, marketing automation, customer relationship management etc. So far SOA is popular for non real-time applications such as accounting, finance and CRM, but it is also increasingly being used for the telecom industry. Multimedia is one of the important services offered in this domain and there is a need for integrating multimedia applications based on SOA.

Web services are easier to configure and deploy than the traditional real time transport protocols and provide a standard way of communications. Hence it provides multimedia applications a secure, reliable and interoperable medium to communicate real time data. In addition, the approach is much more applicable to the multimedia applications where the security and reliability of data is equally or more important than its real time effectiveness e.g. voice recording applications or voicemail services.

Web services platform brings its own challenges when dealing with multimedia or real time data which includes communication overhead, communication delays, format handling etc. Communication delays and overheads are variable and affected by the packet size and frame rate of the data. In the picture of today’s communication networks, data transfer speed and bandwidth, these overheads can have an almost negligible affect depending on the nature of data. In business critical use cases communication overheads are easier to tolerate as compared to traditional real time communications.

The Web services based platform offers a secure and reliable data channel to real time application to stream data whose reliability, accuracy and security have more value and importance than its real time effectiveness.

References

[1] Robert Cailliau, James Gillies, *How the Web Was Born: The Story of the World Wide Web*, ISBN 978-0-19-286207-5, Oxford University Press (Jan 1, 2000)

[2] O'Reilly Media, Inc.2009 The community which introduced the web 2.0 as standard Available from http://oreilly.com/web2/archive/what-is-web-20.html

[3] W3C World Wide Web Consortium, The World Wide Web Consortium (W3C) is an International community that develops standards to ensure the long-term growth of the Web. http://www.w3.org/2002/ws/

[4] RFC 2616, June 1999, Hyper Text Transfer protocol R. Fielding Request for Comments: 2616. UC Irvine, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee http://www.w3.org/Protocols/rfc2616/rfc2616.html

[5] W3C World Wide Web Consortium, The World Wide Web Consortium (W3C) is an International community that develops standards to ensure the long-term growth of the Web. http://www.w3schools.com/xmL/

[6]] W3C World Wide Web Consortium, The World Wide Web Consortium (W3C) is an International community that develops standards to ensure the long-term growth of the Web. http://www.w3schools.com/SOAP/soap_syntax.asp

[7] W3C World Wide Web Consortium, The World Wide Web Consortium (W3C) is an International community that develops standards to ensure the long-term growth of the Web. http://www.w3schools.com/wsdl/

[8] Phillip A. Laplante Real-time systems design and analysis

[9] RFC1889 - RTP: A Transport Protocol for Real-Time Applications, January 1996, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson http://www.faqs.org/rfcs/rfc1889.html

[10] Clark, M.P. (2003). Data Networks IP and the Internet, 1st ed. West Sussex, England: John Wiley & Sons Ltd.

[11] Postel, J. (August 1980). RFC 768: User Datagram Protocol. Internet Engineering Task Force. Retrieved from http://tools.ietf.org/html/rfc768

[12] RFC 793: Transmission Control Protocol. (September 1981). Internet Engineering Task Force. Retrieved from http://tools.ietf.org/html/rfc768

[13] RFC2616: Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, June 1999. This RFC obsoletes RFC 2068.

[14] Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000 http://www.w3.org/TR/2000/NOTE-SOAP-20000508

[15] W3C World Wide Web Consortium, The World Wide Web Consortium (W3C) is an International community that develops standards to ensure the long-term growth of the Web http://www.w3.org/TR/2002/WD-ws-arch-20021114/

9 APPENDIX 1: EXAMPLE WSDL

Following is an example WSDL file which shows how to implement the SOAP based RTP service.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions
  name="SOAP_RTP"
  targetNamespace="http://example.com/rpc/soaprtp"
   xmlns:w="http://example.com/rpc/soaprtp"
  xmlns:p="http://example.com/rpc/soaprtp"
  xmlns:m="http://example.com/rpc/soaprtp_schema"
  xmlns:t="http://example.com/rpc/soaprtp_schema"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
        <s:schema
          elementFormDefault="qualified"
          targetNamespace="http://example.com/rpc/soaprtp_schema"
        xmlns:m="http://example.com/rpc/soaprtp_schema"
        xmlns:t="http://example.com/rpc/soaprtp_schema"
          xmlns="http://www.w3.org/2001/XMLSchema">

          <!-- Define data types -->
      <complexType name="CSRCidentifiers">
        <sequence>
          <element name="CSRCIdentifier" type="xs:string" nillable="false"
minOccurs="1"/>
        </sequence>
      </complexType>

      <complexType name="SenderInfo">
            <sequence>
          <element name="SenderSSRC" type="xs:string" nillable="false"
minOccurs="1"/>
          <element name="NTPTimeStamp" type="xs:string" nillable="true"
minOccurs="0"/>
          <element name="RTPTimeStamp" type="xs:string" nillable="true"
minOccurs="0"/>
          <element name="PacketCount" type="xs:int" nillable="true"
minOccurs="0"/>
          <element name="OctetCount" type="xs:int" nillable="true"
minOccurs="0"/>
        </sequence>
      </complexType>

      <complexType name="ReportBlock">
        <sequence>
          <element name="SSRC" type="xs:string" nillable="false" minOccurs="1"/>
          <element name="FL" type="xs:string" nillable="false" minOccurs="1"/>
          <element name="CNPL" type="xs:string" nillable="false" minOccurs="1"/>
          <element name="EHSNR" type="xs:string" nillable="false"
minOccurs="1"/>
          <element name="IAJ" type="xs:string" nillable="false" minOccurs="1"/>
          <element name="LSR" type="xs:string" nillable="false" minOccurs="1"/>
          <element name="DLSR" type="xs:string" nillable="false" minOccurs="1"/>
        </sequence>
      </complexType>

      <complexType name="ReportBlocks">
```

```
        <sequence>
          <element name="ReportBlock" type="t:ReportBlock"/>
        </sequence>
      </complexType>

      <complexType name="Extension">
        <sequence>
          <element name="ID" type="xs:string" nillable="false" minOccurs="1"/>
          <element name="Header" type="xs:string" nillable="false"
minOccurs="1"/>
        </sequence>
      </complexType>

      <complexType name="Extensions">
        <sequence>
          <element name="Extension" type="t:Extension"/>
        </sequence>
      </complexType>

      <complexType name="RTPHeader">
        <sequence>
          <element name="Version" type="xs:string" nillable="false"
minOccurs="1"/>
          <element name="SequenceNumber" type="xs:string" nillable="false"
minOccurs="1"/>
          <element name="SSRCIdentifier" type="xs:string" nillable="false"
minOccurs="1"/>
          <element name="CSRCidentifiers" type="t:CSRCidentifiers"/>
          <element name="Extensions" type="t:Extensions"/>
        </sequence>
      </complexType>

      <complexType name="RTCPHeader">
        <sequence>
          <element name="Version" type="xs:string" nillable="false"
minOccurs="1"/>
          <element name="SenderInfo" type="t:SenderInfo"/>
          <element name="ReportBlocks" type="t:ReportBlocks"/>
          <element name="Extensions" type="t:Extensions"/>
        </sequence>
      </complexType>


      <!-- Define message data types -->
      <element name="SOARTPDataPacket">
        <complexType>
          <sequence>
            <element name="Header" type="t:RTPHeader"/>
            <element name="Payload" type="xs:string" nillable="false"
minOccurs="1"/>
          </sequence>
        </complexType>
      </element>

      <element name="SOARTCPSR">
        <complexType>
          <sequence>
            <element name="Header" type="t:RTCPHeader"/>
          </sequence>
        </complexType>
      </element>

      <element name="SOARTCPRR">
        <complexType>
          <sequence>
            <element name="Header" type="t:RTCPHeader"/>
          </sequence>
        </complexType>
      </element>

    </types>

  <!-- Define messages -->
   <message name="SOARTPDataPacket">
     <part name="parameters" element="m:SOARTPDataPacket" />
   </message>

   <message name="SOARTCPSR">
     <part name="parameters" element="m:SOARTCPSR" />
   </message>

   <message name="SOARTCPRR">
     <part name="parameters" element="m:SOARTCPRR" />
   </message>

   <message name="Response">
        <part name="parameters" element="m:Response" />
  </message>

  <!-- Define port and its operations -->
  <portType name="RTP_SoapPort">

        <operation name="SOARTPDataPacket">
          <input message="p:SOARTPDataPacket" />
          <output message="p:Response" />
        </operation>

        <operation name="SOARTCPSR">
          <input message="p:SOARTCPSR" />
          <output message="p:Response" />
        </operation>

        <operation name="SOARTCPRR">
          <input message="p:SOARTCPRR" />
          <output message="p:Response" />
        </operation>

   </portType>

  <!-- Declare bindings -->

  <binding name="RTP_SoapBinding" type="p:RTP_SoapPort" >

        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />

        <operation name="SOARTPDataPacket">
          <soap:operation soapAction="http://example.com/rpc/soaprtp"
style="document" />
          <input>
         <soap:header part="SOAPRTPHeader" message="p:SOARTPDataPacket"
use="literal" />
            <soap:body use="literal" parts="parameters" />
          </input>
          <output>
            <soap:body use="literal" />
          </output>
        </operation>

      <operation name="SOARTCPSR">
        <soap:operation soapAction="http://example.com/rpc/soaprtp"
style="document" />
       <input>
         <soap:header part="SOAPRTPHeader" message="p:SOARTCPSR" use="literal" />
         <soap:body use="literal" parts="parameters" />
       </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>

    <operation name="SOARTCPRR">
      <soap:operation soapAction="http://example.com/rpc/soaprtp"
style="document" />
      <input>
        <soap:header part="SOAPRTPHeader" message="p:SOARTCPRR" use="literal" />
        <soap:body use="literal" parts="parameters" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>

  </binding>

 <!-- Declare services -->

 <service name="SOAP_RTP" >
      <port name="RTP_SoapPort" binding="w:RTP_SoapBinding" >
        <soap:address location="http://localhost/rpc/soaprtp.asmx" />
      </port>
 </service>

</definitions>
```