Network Protocol Design – Voice Chat

Group 14: Bill Shaya, Jordan Osecki, Bob Cochran

CS544: Computer Networks, Professor Kain

February 11, 2009 – Winter Term 2008-2009

**Voice Chat Network Protocol Design Document**

# Overview – Formalize this some, add further features descriptions

-Description of the service that protocol and server will provide

-Describe basic service and options/conditions available

Building from what we learned during protocol analysis of IRC, our group wishes to design and implement a new protocol that provides a platform for voice chat across TCP/IP networks. The goal of the protocol is to enable real-time voice chat, conferencing, and announcement broadcast across the Internet that takes advantage of a simple connection model and distributed server architecture to enable tens, hundreds, or even thousands of simultaneous participants in a single conversation.

During our analysis, we observed that many of the IRC concepts could be reused to provide a mechanism for voice chat. Some of the concepts that we would like to reuse as building blocks for our protocol include the server connection model, joining and parting from a channel, and distributing the servers to support many users. We also plan to start with the IRC DFAs from the analysis paper in developing the voice chat DFAs.

There are also various important distinctions for our new protocol. These include the following:

* The use of a TCP port for control messages and a UDP port for voice transfer.
* Definition of a new control message set that targets voice chat, utilizes XML for message encapsulation, and is easily extendable.
* An objected oriented approach to our message definition that allows for a simple mapping (serialization) between the message set and the objects within our client and server software.
* Definition of a compact voice packet, which may be modeled after RTP, for the transfer of voice packets utilizing UDP. The packet header will be optimized for real-time transfer and consist of concise bit fields to define parameters such as sequence number and payload length.

# Implementation Plan

* Implement a subset of the protocol to demonstrate basic voice connectivity and transfer (e.g., two party chat). We do not plan to implement the functionality required for multiple, distributed servers.
* The client will be implemented in Java, utilize a GUI, and be tested on at least Microsoft Windows.
* The server will be implemented in C/C++ and target a Linux host.
* An XML package / library will be utilized for parsing the XML messages.
* The voice packets will support only G.711 (64 kbit/s) telephony grade voice.

# Risks

1. It is our hope to be able to demonstrate live voice with a microphone and speaker; however, the use of voice files will be a fallback position.
2. The voice jitter buffer will be simple, and we may not have time to support complex functions like packet reordering.
3. Achieving real-time performance out of a generic, desktop Linux system may be difficult. It may be necessary to insert long delays in multiple areas in order to demonstrate continuous voice. Another option is to demonstrate the voice chat system utilizing an embedded Linux system. This will be explored during development.

# System Architecture

The figure below depicts at a high level the distributed nature of the voice chat system. Each server supports the familiar user commands for joining a chat (e.g., NICK, USER, JOIN, PART, etc.). This provides for a simple and friendly mechanism for channel creation and call control. The server implements in software a separate summation / conference circuit per user that sums, on a sample-by-sample basis, each of the incoming user streams that the user has chosen to not ignore. The input streams include the conferences from other servers on the network. Note that the user may ignore individual streams that are local to its server but cannot separate individual streams from remote servers[[1]](#footnote-2).

As an example, the figure below shows six clients joined together in a conference call on a created channel distributed across 4 servers (A..D). Server A implements three conference circuits: Client 1 receives a summation of client 2 and Server B. Client 2 receives a summation of client 1 and Server B. Server B receives a summation of clients 1 and 2. Server C, which does not have any clients simply relays Server B’s stream to Server D and vice versa.



Figure 1: Distributed Voice Chat Server Architecture

Considering that the payload for voice chat is digitized voice rather than ASCII strings, the voice packet utilizes well defined bit fields that specify various parameters including voice activity, sequence number, and payload length.

# Messages - XML Representation and Voice Representation

-Define how the messages will be delineated

-Define control and data messages

-Show all pieces of the messages including data types, enumerations, etc.

-Probably keep the diagram but move the rest to an appendix

**XML Control Message Commands**

Our protocol proposes using an XML representation via TCP/IP for control messages. Below, in Figure 2, is a sample XSD file (graphical) which defines the schema of the XML hierarchy. As shown, we have a high level root node "VoiceIRC". Child nodes break down into the various class representations. For example, the graphic belows shows the model of "Channel" and "User". Under the "Channel" node, we have nodes for "ChannelName", "ChannelTopic", and "ChannelMode". In addition, we show a "User" node which contains sub nodes for "UserName", "IPAddress", and "UserMode". Another important thing that is needed at the high level node is the "CommandName" because when sending and receiving XML, the software needs to know exactly what command or response the client is initiating, and based on that, the protocol will dictate extract which nodes under the tree to parse or create.

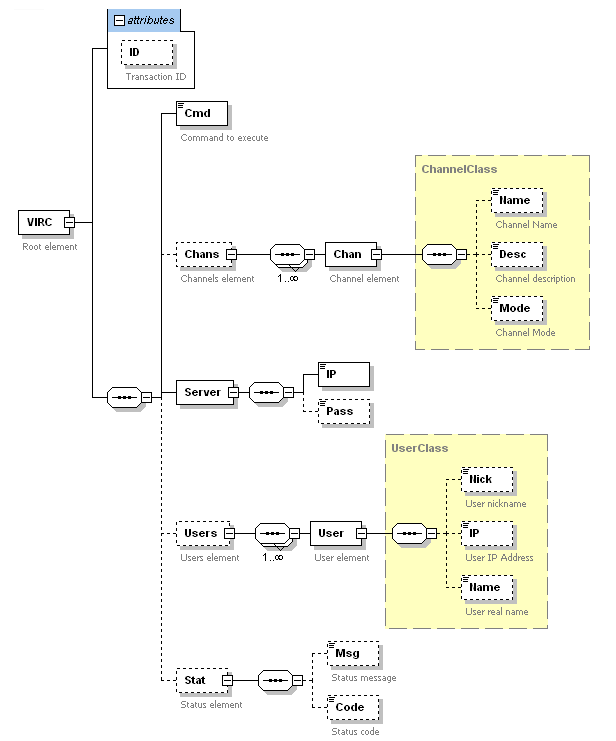


Figure 2: Sample XSD In Graphical Representation

Commands initiated by the client are displayed below. Note that all commands have an attribute “ID” which is used as a transaction ID. The client will increment this transaction ID with each command sent to the server. The server will process the command and return a response with the same transaction ID so that the client can match the response to the command.

The client will be designed as a Java GUI which supports a simple interactive design in order to support the range of commands listed below. The typical client concept of operation includes the following interactivity:

1. The client issues a “Conn” command to connect to a remote server
2. The client issues a “GetChans” command to query a list of existing channels hosted on the server
3. The client issues a “Join” command to enter an existing channel or create a new channel
4. Upon joining the channel, the user will either issue a “GetUsers” command or the server will automatically push the data down to the client such that the GUI displays all users in the conference.
5. Upon entering a conference, voice will be enabled such that the user can send voice data to the server, which will handle all processing and distribute the samples accordingly.
6. If the user is the first client into the conference, he is automatically promoted to an operator privilege, and may issue a “Kick” command to remove any other clients, a “Ban” command to remove and permanently bar another client from the channel, a “Mute” command to no longer hear what another client is saying in a certain channel, and a “NewDesc” command to change the title and/or description of a channel.
7. Any time during the conference, the user may issue a “Part” command to exit the conference.
8. When the user wishes to disconnect from the server, he initiates a “Disconn” command.

**Conn**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>Conn</Cmd>

<Server>

<Pass>Optional Password</Pass>

</Server>

<Users>

<User>

<Nick>Bill</Nick>

<IP>192.168.1.1</IP>

<Name>Bill Shaya</Name>

</User>

</Users>

</VIRC>

***Response: Default Command Response with ‘0’ for success, ‘1’ for failure, and an optional message to describe the result***

**Disconn**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>Disconn</Cmd>

<Users>

<User>

<Nick>Bill</Nick>

</User>

</Users>

</VIRC>

***Response: Default Command Response with ‘0’ for success, ‘1’ for failure, and an optional message to describe the result***

**Join**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>Join</Cmd>

<Chans>

<Chan>

<Name>C# Development</Name>

</Chan>

</Chans>

<Users>

<User>

<Nick>Bill</Nick>

</User>

</Users>

</VIRC>

***Response: Default Command Response with ‘0’ for success, ‘1’ for failure, and an optional message to describe the result***

**Part**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>Part</Cmd>

<Chans>

<Chan>

<Name>C# Development</Name>

</Chan>

</Chans>

<Users>

<User>

<Nick>Bill</Nick>

</User>

</Users>

</VIRC>

***Response: Default Command Response with ‘0’ for success, ‘1’ for failure, and an optional message to describe the result***

**Kick**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>Kick</Cmd>

<Chans>

<Chan>

<Name>C# Development</Name>

</Chan>

</Chans>

<Users>

<User>

<Nick>Bill</Nick>

</User>

</Users>

</VIRC>

***Response: Default Command Response with ‘0’ for success, ‘1’ for failure, and an optional message to describe the result***

**Ban**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>Ban</Cmd>

<Chans>

<Chan>

<Name>C# Development</Name>

</Chan>

</Chans>

<Users>

<User>

<Nick>Bill</Nick>

</User>

</Users>

</VIRC>

***Response: Default Command Response with ‘0’ for success, ‘1’ for failure, and an optional message to describe the result***

**Mute**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>Mute</Cmd>

<Chans>

<Chan>

<Name>C# Development</Name>

</Chan>

</Chans>

<Users>

<User>

<Nick>Bill</Nick>

</User>

</Users>

</VIRC>

***Response: Default Command Response with ‘0’ for success, ‘1’ for failure, and an optional message to describe the result***

**NewDesc**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>NewDesc</Cmd>

<Chans>

<Chan>

<Name>C# Development</Name>

<Desc>This is the place to learn C#</Desc>

</Chan>

</Chans>

</VIRC>

***Response: Default Command Response with ‘0’ for success, ‘1’ for failure, and an optional message to describe the result***

**GetChans**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>GetChans</Cmd>

</VIRC>

***Response:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>GetChans</Cmd>

<Chans>

<Chan>

<Name>C# Development</Name>

<Desc>This is the place to learn C#</Desc>

<Mode>public</Mode>

</Chan>

<Chan>

<Name>Java Development</Name>

<Desc>This is the place to learn Java</Desc>

<Mode>public</Mode>

</Chan>

</Chans>

<Stat>

<Msg>Some optional text message here</Msg>

<Code>0</Code>

</Stat>

</VIRC>

**GetUsers**

***Command:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>GetUsers</Cmd>

</VIRC>

***Response:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>GetUsers</Cmd>

<Chans>

<Chan>

<Name>Java Development</Name>

</Chan>

</Chans>

<Users>

<User>

<Nick>Bill</Nick>

</User>

<User>

<Nick>Bob</Nick>

</User>

</Users>

<Stat>

<Msg>Some optional text message here</Msg>

<Code>0</Code>

</Stat>

</VIRC>

**Default Command Response**

***Response for success:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>Join</Cmd>

<Stat>

<Msg>Some optional text message here</Msg>

<Code>0</Code>

</Stat>

</VIRC>

***Response for failure:***

<VIRC ID="0" xsi:schemaLocation="http://www.drexel.edu IRC.xsd" xmlns="http://www.drexel.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Cmd>Join</Cmd>

<Stat>

<Msg>Some optional text message here</Msg>

<Code>1</Code>

</Stat>

</VIRC>

**Full Schema – (Move to appendix eventually)**

<?xml version="1.0" encoding="UTF-8"?>

<!-- edited with XMLSpy v2008 sp1 (http://www.altova.com) by L-3 Communications (L-3 Communications) -->

<xs:schema xmlns="http://www.drexel.edu" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.drexel.edu" elementFormDefault="qualified" attributeFormDefault="unqualified">

<xs:element name="VIRC">

<xs:annotation>

<xs:documentation>Root element</xs:documentation>

</xs:annotation>

<xs:complexType>

<xs:sequence>

<xs:element name="Cmd">

<xs:annotation>

<xs:documentation>Command to execute</xs:documentation>

</xs:annotation>

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="Conn"/>

<xs:enumeration value="Disconn"/>

<xs:enumeration value="Join"/>

<xs:enumeration value="Part"/>

<xs:enumeration value="Kick"/>

<xs:enumeration value="Ban"/>

<xs:enumeration value="Mute"/>

<xs:enumeration value="NewDesc"/>

<xs:enumeration value="GetChans"/>

<xs:enumeration value="GetUsers"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="Chans" minOccurs="0">

<xs:annotation>

<xs:documentation>Channels element</xs:documentation>

</xs:annotation>

<xs:complexType>

<xs:sequence maxOccurs="unbounded">

<xs:element name="Chan" type="ChannelClass">

<xs:annotation>

<xs:documentation>Channel element</xs:documentation>

</xs:annotation>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="Server">

<xs:complexType>

<xs:sequence>

<xs:element name="IP" type="xs:string"/>

<xs:element name="Pass" type="xs:string" minOccurs="0"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="Users" minOccurs="0">

<xs:annotation>

<xs:documentation>Users element</xs:documentation>

</xs:annotation>

<xs:complexType>

<xs:sequence maxOccurs="unbounded">

<xs:element name="User" type="UserClass">

<xs:annotation>

<xs:documentation>User element</xs:documentation>

</xs:annotation>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="Stat" minOccurs="0">

<xs:annotation>

<xs:documentation>Status element</xs:documentation>

</xs:annotation>

<xs:complexType>

<xs:sequence>

<xs:element name="Msg" type="xs:string" minOccurs="0">

<xs:annotation>

<xs:documentation>Status message</xs:documentation>

</xs:annotation>

</xs:element>

<xs:element name="Code" type="xs:unsignedInt" minOccurs="0">

<xs:annotation>

<xs:documentation>Status code</xs:documentation>

</xs:annotation>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

<xs:attribute name="ID" type="xs:unsignedInt">

<xs:annotation>

<xs:documentation>Transaction ID</xs:documentation>

</xs:annotation>

</xs:attribute>

</xs:complexType>

</xs:element>

<xs:complexType name="ChannelClass">

<xs:sequence>

<xs:element name="Name" minOccurs="0">

<xs:annotation>

<xs:documentation>Channel Name</xs:documentation>

</xs:annotation>

<xs:simpleType>

<xs:restriction base="xs:string"/>

</xs:simpleType>

</xs:element>

<xs:element name="Desc" type="xs:string" minOccurs="0">

<xs:annotation>

<xs:documentation>Channel description</xs:documentation>

</xs:annotation>

</xs:element>

<xs:element name="Mode" minOccurs="0">

<xs:annotation>

<xs:documentation>Channel Mode</xs:documentation>

</xs:annotation>

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="public"/>

<xs:enumeration value="private"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:sequence>

</xs:complexType>

<xs:complexType name="UserClass">

<xs:sequence>

<xs:element name="Nick" minOccurs="0">

<xs:annotation>

<xs:documentation>User nickname</xs:documentation>

</xs:annotation>

<xs:simpleType>

<xs:restriction base="xs:string"/>

</xs:simpleType>

</xs:element>

<xs:element name="IP" type="xs:string" minOccurs="0">

<xs:annotation>

<xs:documentation>User IP Address</xs:documentation>

</xs:annotation>

</xs:element>

<xs:element name="Name" type="xs:string" minOccurs="0">

<xs:annotation>

<xs:documentation>User real name</xs:documentation>

</xs:annotation>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:schema>

# Deterministic Finite Automata – Use IRC One

-Define and show it, at least 4 states, 10 unique messages (only 2 error msgs)

-Describe it’s statefulness

# Extensibility – Bob talk about voice messages, Bill/Jordan talk about XML

-Describe how it allows future extensibility

-Version info, handshake negotiation, etc.

# Performance – Bob performance of Voice, etc., QOS

-Also is supposed to be expressed

# Security – Bob talk about Encryption, Bill/Jordan talk about Login

-Describe how it will ensure right people can use it, allow everyone?

-Describe all authentication mechanisms (not all need to be implemented, can be hardcoded)

-How do you ensure it is always working correctly?

\*\*\*Will we need a bibliography?

1. The user can either listen to all users or none from remote servers. [↑](#footnote-ref-2)