



San José State
UNIVERSITY

Project Report

Extensible Messaging and Presence Protocol (XMPP)

By
“Team 14”

Under the guidance of:
Prof. Shai Silberman
CMPE 208-01, Spring 2016
San Jose State University

Team members

Omkar Rege
Ankur Sheth
Girish Bangalore Ramesh
Avinash Closepet Suresh

Abstract

The project report focuses on Extensible Messaging and Presence Protocol (**XMPP**). XMPP is a communication protocol with open standard for messaging and presence. It enables real time transfer of data between two network entities. It's a language independent software for every platform and libraries. This provides basic presence, message and routing features. This open standard is suitable for all latest technology like IOT, smart grid, WebRTC and networking. XMPP is robust and powerful as its architecture is decentralized. It provides near real time communications through text, audio and video. Many social networking platforms uses XMPP such as Facebook, what's app, Gtalk and yahoo. The Core XMPP is explained in [RFC 6120] and XMPP message format is explained in [RFC6122]. XMPP is one of the continuously improving standards. In order to perform the XMPP Lab, we have used OpenFire (version 4.0.2) server and Spark (version 2.7.7) IM client so as to study and generate the XMPP traffic.

Contribution of each team member

Student ID	Team Member	Contribution
010120240	Girish Bangalore Ramesh	Abstract, Introduction, Overview
010739924	Ankur Sheth	Lab environment setup, Lab Execution, Observations
010720398	Omkar Rege	Observations, Packet Captures
010129483	Avinash Closepet Suresh	Architecture, Standards, Conclusion, References

Although the above table shows individual team member's contribution in performing this lab, every member has tried to put in his efforts equally in completing the lab and drafting the lab report.

Table of Contents

Abstract.....	2
Contribution of each team member	2
Introduction	4
Motivation for XMPP	7
Overview	7
XMPP Architecture	7
XMPP Implementation	13
Lab Environment setup	15
Captured XMPP packets.....	23
Conclusion.....	31
References	32

Introduction

XMPP is a communication protocol with open standard for messaging and presence. It enables real time transfer of data between two network entities. It's a language independent software for every platform and libraries. XMPP is basically designed for instant messaging and information presentation.

Now XMPP can support for Voice over IP, message oriented middleware and also file transfer. Current applications use XMPP extensible.

XMPP is an open standard by IETF and its been continuously extended by the standard foundation.

The main features of XMPP are as follows:

- It's a decentralized network. We can set up a XMPP server and have their clients and services connected to it.
- XMPP server and client implementation are readily available by many open source contributors and there are many plugins available for providing various gateway services.
- New messages and services can be easily extended in the XMPP
- Custom extensions are also possible.

XMPP can be categorized for two:

1. Services
2. Application

Services:

This is a kind of functionality which can be incorporated by other applications. Few of the services are as follows:

- Channel Encryption
- Peer-to-peer media sessions
- Authentication
- Presence
- Contact List
- Notifications
- One-to-one messaging
- Multi-party messaging

Detailed explanation about services is as below:

Channel encryption: To build a secure connection channel it's important that the client-server or the server-server is encrypted.

Authentication: A clients trying to connect to the server must have an authentication system. It's always safe to have a secure connection.

Presence: This is used to find other entities in a given area. This gives information about status of the user.

Contact lists: This service is used to maintain a list of entities on a server. One-to-one messaging Instant messaging applications uses this service to send and receive messages between them.

Notifications: Notifications are instant alert messages on any entity.

Service discovery: This can be used to find the services provided by other entities and to find the services used by others.

Peer-to-peer: This manages the services like instant messaging, video chat etc. connection between other entities.

Applications:

We can use the core XMPP services to build custom applications like:

- System control
- Group chat
- Gaming
- Instant messaging
- Geo Location
- Middleware and cloud computing

Below is a list of applications built using XMPP:

Instant messaging: The core services like Contact list, presence and one-to-one messaging can be used to build an application using XMPP.

Geo-location: Maps or tracking can be integrated using the notification service by having geo-location as payload in XMPP.

Cloud computing: This XMPP service can be used for lightweight communication, calculation, and cloud management.

Voice over IP: Jingle which is the XMPP media session is used here. Google talk used XMPP for VOIP for the first time.

Identity services: Jabber ID can be used for identifying services. These services like Jabber can use XMPP in building their identity.

Gaming: One-to-one messaging and multi-party messaging and extensions are the core components used to build gaming application.

Types of Messages:

Chat: One-to-one conversation.

Error: If any error has occurred w.r.t the previous messages.

Group chat: It's the messages in multiuser chat application.

Headline: This is just a broadcast kind off messages

Normal: This is the message used in the one-to-one conversation

Communication Primitives:

The client and the server which is decentralized communicates with each other with the help of XML data streams. The streams basic unit is the stanza. Each and every stanza are built with the help of XML codes with the existing libraries. The three main component of the stanza are message, presence and IQ (information and query). To get the data from one place to other place by using a push method. A message stanza looks like the example shown below.

Message:

```
<message from=abc@gmail.com/xyz
          to=pqr@jabber.org
          type=" chat">
<body>How are you? </body>
<subject>Query</subject>
</message>
```

In the above message stanza format there are two important things to consider, they are the "TO" and "FROM" message address and their respective id. The client side does not contain the "FROM" message, but it is specified in the server end in order to avoid spamming the address. The payload includes the "body" stanza and the "subject" that are used while chatting.

Presence:

```
<presence from=abc@gmail.com/xyz>
  <show>do not disturb</show>
  <status>in a meeting</status>
</presence>
```

This type of the message is used in the instant messaging application, which uses the status message which displays the status of the person in the contact. Whenever the user of this application is online, the client side software will display the status on the server side. This mainly helps in sending a notification to the contacts that are in the users list, and also helps in displaying the status of the user and also display the presence in the interface.

IQ:

```
<iq from="abc@gmail.com/wz2"
  id="rr82a1z7"
  to="sankur91@gmail.com"
  type="get">
<query xmlns="jabber:iq:roster"/>
</iq>
```

IQ is a like request and response mechanism in HTTP. IQ enables one entity to query other entity and get response other entity. It can have various types like get, result, set and error.

Motivation for XMPP

As we know instant messaging (chat) applications such as Google Talk, WhatsApp, Cisco WebEx and others have been gaining popularity all the way. A deep look into those applications reveals the usage of XMPP protocol for messaging.

A deep dig into the XMPP protocol, we can explore few interesting features which are as below:

- Open Standards- Approved instant messaging and are free to implement.
- Security-These can be isolated and the messages will be encrypted.
- Flexibility- We can customize the protocol as per our needs.
- Decentralization-The architecture is very much similar to email and there is no need for master server.

As being much interested in developing network applications such as WhatsApp, it would be very helpful for us to learn those protocols and its usage in such similar applications.

We have to learn XMPP for many of the following reasons:

To build any real time application.

To understand network topology and how the XMPP based messages travel via Internet.

Developing Middleware applications or trying for middleware services.

Overview

The project report focuses on Extensible Messaging and Presence Protocol (XMPP). In order to perform the XMPP Lab, we have used OpenFire (version 4.0.2) server and Spark (version 2.7.7) IM client so as to study and generate the XMPP traffic. The Packets are captured and analyzed using WIRESHARK, a network analyzing tool. The report explains about the lab environment setup required to perform XMPP lab using Linux, Wireshark, Openfire server and Spark IM Client Environment, it also briefly introduces XMPP. The report describes the various steps involved in generation of XMPP traffic, XMPP packet observations and analysis of captured XMPP packets, it also depicts different graphs and statistics received from Wireshark during the lab execution. The report concludes with Pros and cons of XMPP, XMPP standards, and the few learnings from this project.

XMPP Architecture

XMPP stands for extensible messaging and presence protocol, this consists of open technologies for instant messaging, voice and video calls, middleware and also generalized routing of XML data. XMPP was developed by an open source community called as jabber to provide an open and decentralized instant messaging services.

The architecture of XMPP is similar to that of SMTP protocol and DNS (Domain Name System). The architecture is decentralized, meaning the control is distributed to many parts. The decentralization of the system increases the reliability and scalability. A simple XMPP consists of a client with a name that's unique which communicates with another client which has a unique name through a router. The server is mainly used for routing; in this case each client is considered to be a part of some domain.



Figure: Simple XMPP architecture

There is a more complex architecture as well. Servers are mainly used for the routing purpose between the domains that is from one client to another client. This architecture also includes gateways for the translation of message from a foreign domain and protocols. The block diagram below shows a XMPP network with gateways to a short message services (SMS) and also SMTP domains. The main idea behind the XMPP is to provide universal connectivity among different endpoint protocols. This has made XMPP a universal backbone protocol. The termination of client to server is all controlled by the gateway, and also initiates a new session to the target endpoint protocol.

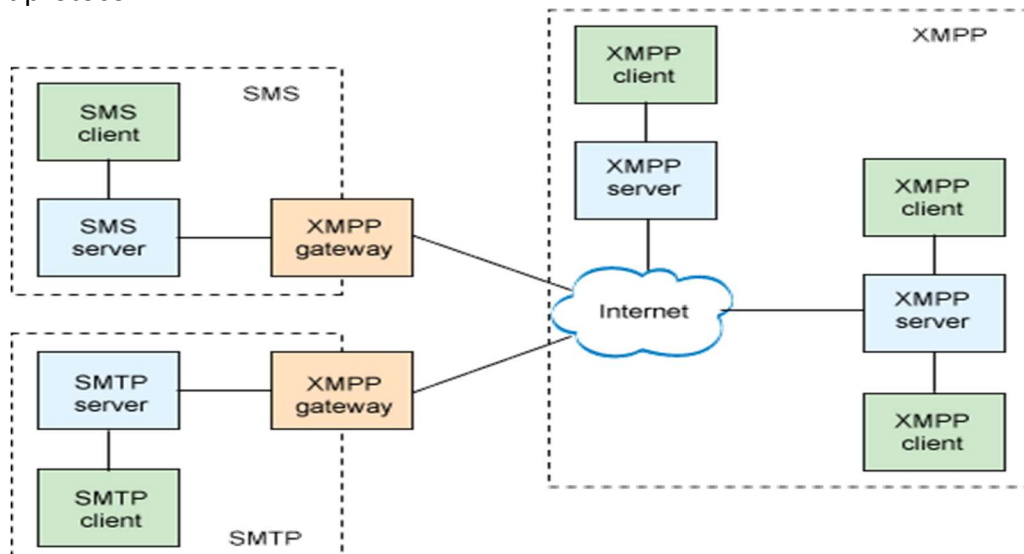


Figure: Complex XMPP architecture

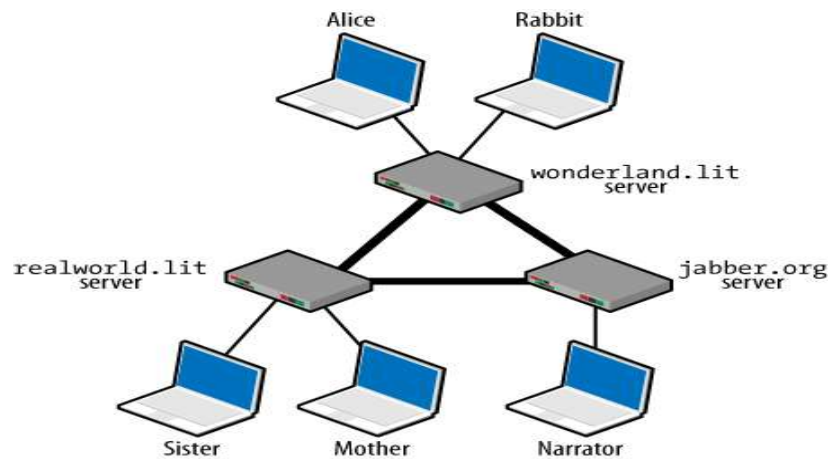
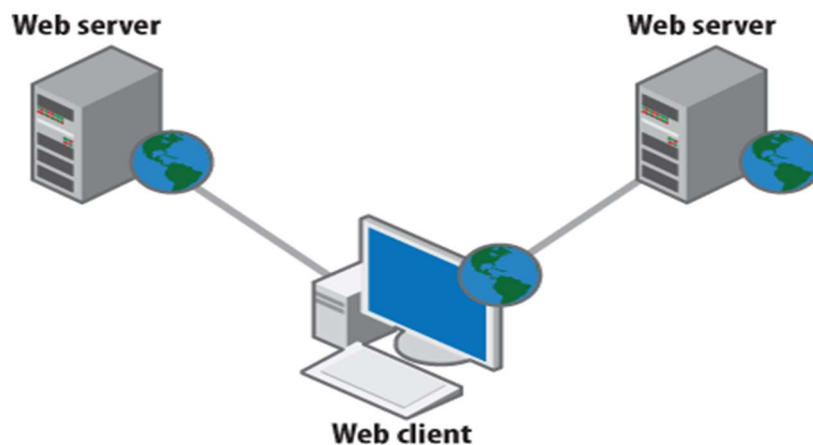


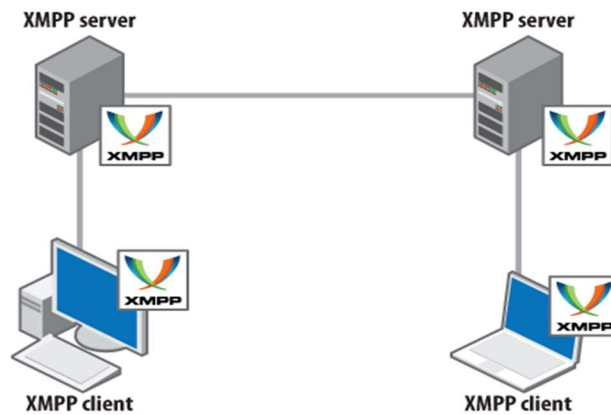
Figure: Example of XMPP architecture

The community of XMPP have always tried to simplify the client structure, by making the servers more complex, this has in turn lead to a great innovation in the field of network technology. It is very important in understanding the architectural differences between SMTP, Jabber and web. Whenever a user browses the web, it does not connect to the server directly, but it uses HTML which in turn refers to the server to load the requested information. As a result, the web based communication is not inter-domain communication. Which is in other words termed as the federation. The below diagram shows the federation.



The major distinction between the XMPP and web is that, XMPP has inter domain communications. That is whenever the client or the user sends an XMPP request to any other contact in a different domain, it firsts connects to the home server. The client here connects to the server directly without any HOPS in the middle. This technique is very advantageous in protecting the user from address spoofing and any other malicious activities.

The below figure shows the gives an insight to the basic difference between the web architecture associated with lots of clients and servers along with servers that are interconnected in one HOP.



Address Format for XMPP:

The XMPP address or jabber ID's (JID's) are similar to that of standard e-mail address with some notable changes. JID's have node (optional), a domain and a resource in the form which is again optional. This can be represented as shown below.

[node "@" domain ["/" resource]]

The XMPP address depends on the DNS (Domain Name System) and not on the IP address. This will help us to use the XMPP very easily without trying to memories the IP address. Here a user using XMPP can login to server multiple times based on the requirements. The main functionality of the resource is to denote the location. For example a user can have a jabber id for his main terminal to be Avinash.closepet@nationalgeography.channel.guv/terminal.

```

jid      = [ localpart "@" ] domainpart [ "/" resourcepart ]
localpart = 1*(nodepoint)
;
; a "nodepoint" is a UTF-8 encoded Unicode code
; point that satisfies the Nodeprep profile of
; stringprep
;
domainpart = IP-literal / IPv4address / ifqdn
;
; the "IPv4address" and "IP-literal" rules are
; defined in RFC 3986, and the first-match-wins
; (a.k.a. "greedy") algorithm described in RFC
; 3986 applies to the matching process
;
; note well that reuse of the IP-literal rule
; from RFC 3986 implies that IPv6 addresses are
; enclosed in square brackets (i.e., beginning
; with '[' and ending with ']'), which was not
; the case in RFC 3920
;
ifqdn     = 1*(namepoint)
;
; a "namepoint" is a UTF-8 encoded Unicode
; code point that satisfies the Nameprep
; profile of stringprep
;
resourcepart = 1*(resourcepoint)
;
; a "resourcepoint" is a UTF-8 encoded Unicode
; code point that satisfies the Resourceprep
; profile of stringprep
;
  
```

Figure: XMPP Address format

XMPP Connection Cycle:

The data that are being exchanged is called as a stanza. Mainly there are three types of stanzas they are presence, message and IQ.

Presence: This will tell us whether the user is offline or online or status messages.

Message: this is nothing but the message or the data sent between the two users.

IQ: this provides query based information. Get the user info, update the user info etc.

In order to exchange the data or the stanza we need to establish a stream between a server and a client or a stream between server and a server. Mainly there are four major steps in the life cycle of XMPP. They are Connections, connecting the streams, authentication and disconnection. These four steps are explained in details below with the help of a block diagram.

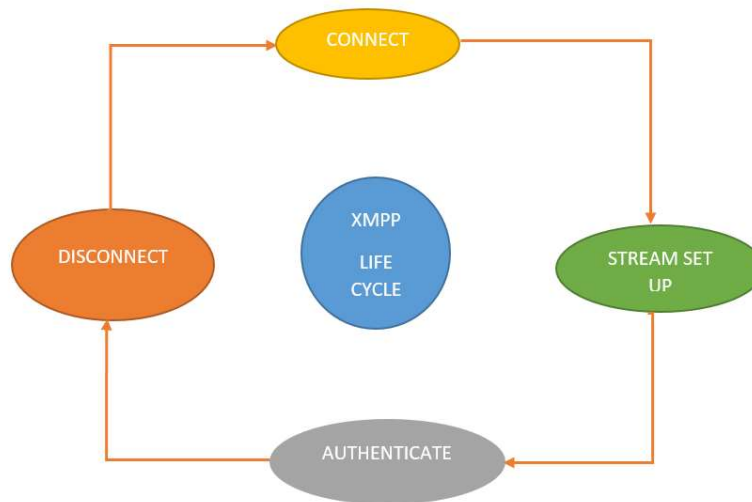


Figure: XMPP Connection cycle

Connection: in order to establish the streams, a server and a client or server and a server should first establish a connection between them. This is the first and the important step in the XMPP life cycle. Normally when we want a page to be viewed we give the website name and the corresponding IP address is stored in the DNS server. This information present in the server is used to load the page. But in case of XMPP, whenever a client wants to establish a connection with the server, it first has to get the appropriate SRV record. SRV record is nothing but the service record. Once this record is obtained by the client, the connection between the client and the server is established.

Creating Streams: Once the connection is established, in order to start the communication, we need to open the streams from both the ends. After this step the server will respond with a tag <stream: stream>. Once the streams are opened on both the sides, the stanzas are now exchanged between the server and the clients. Along with the <stream: stream> tag the server also sends the <stream: features> tag. These tags are mainly used for the authentication and encryption options.

Authentication: After the connection and the establishment of the streams we now have to concentrate on authentication. XMPP uses simple authentication and security layers' protocol

(SASL) to authenticate the clients. Clients bound to a particular resource only after authentication is approved. When we are trying to connect two servers, then process for authenticating is little different. Here the TLS certificates are used or exchanged between servers, the recipient server in turn uses the protocol called as the “dial back” to verify the identity of the senders.

Disconnection: before starting the communication user would have logged into the system. Once the communication is done or the XMPP session is completed, we have to logout to end the connection this process is called as disconnection. In order to disconnect we use this tag `</stream: stream>`. After disconnecting the user will be offline or unavailable for any kind of XMPP communications.

Streaming XML: For the sessions to be started, TCP connection which is long lived must be enabled and the XML streams must be handled. Once this is done server will also open a stream as a result, one stream in both the direction will be present. The stream is nothing but XML document which is piling up in an incremental order over time. Only once the stream is sent to the server, the XML snippets are exchanged. Mainly there are three snippets

`<message/>`

`<presence/>`

`<IQ/>`

Stanza forms the basic fundamental unit. As the stream is sent to the server, we can send unlimited stanzas as per the requirements. In this XMPP design a client can send multiple request to the server and will be waiting for the reply from the server. The servers reply to all the request as soon as they are flooded with the requests.

Channel Encryption: The most important thing in the XMPP is the channel encryption, this mainly uses Transport Layer Security (TLS). This technology is the very much needed technology and compulsory for all the servers and clients that use XMPP. This is used as it overcomes the problem of server identity. This helps in encrypting the traffic on client to server and server to server links, in a straightforward way.

Channel encryption is vulnerable for many attacks, even though we encrypt the message we want to send to our friend or colleague, it remains unencrypted inside the servers, this may lead to a potential risk. Middle man or the server admins can log all our messages as it passes through the network link.

Authentication: For the communication to be safe, authentication is very important. In XMPP there are several methods to provide strong identity on the network. The exact credentials are always checked in the process of login. Authentication process make sure that the person in your contact is the same old person added few weeks, months or even a few years back. From address is stamped by the server, this ensures a trust in the message we receive. As mentioned previously XMPP uses simple authentication and security layers (SASL) protocol for authentication. A handshake of authentication with SASL client includes an `<auth/>` element initially and ends with a `<success/>` element.

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'  
  mechanism='appropriate value here'></auth>  
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

At the end once the <success/> element is received, XML streaming is initialized overran existing TCP connection. If there is any failure, then a response message called <failure/> with the exact reason for failure is sent.

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>  
  <reason for failure here/>  
</failure>
```

These are some of the main reasons for which the failure message will be generated. They are <aborted/>, <account-disabled/>, <credentials-required/>, etc.

Error Handling: If routing an XML stanza to a particular server is unsuccessful, then the server receiving must return an error to the server sending the stanza. If the remote domain name is unsuccessful <remote-server-timeout/> error is returned. But if the server is found and not able to exchange the streams then a <remote-server-timeout/> error stanza is returned. However, a stream exchanged or sent to a intended server is successful but not able to send the stanza to the recipient server, then an appropriate error message must be sent to the sender server.

XMPP Implementation

Instant Messaging [IM]

The initial motive of jabber technology was to deliver a fast messaging platform. Though we consider IM as a person-to-person chat it actually includes sending messages quickly over the network without thinking much about the recipient of the messages. Therefore, XMPP servers are recognized for quick or instant messaging. Suppose user1 at server A sends a IM to user2 at server B. Then first, client of user1 sends message to server A using client-to-server XML stream with no intermediate hops. The server A then captures from and to fields in the stanza without any XML parsing or packet inspection since it consumes lot of time. After seeing that the message is intended for server B the Server A then sends it to the server B by opening server -to server XML stream. Upon receiving the message server B check if user2 is online if yes then it sends it to user2 over server-to-client stream. Note that user1 and user2 should be Presence aware which makes the delivery of messages quick.

During the chat session user requesting the chat sends his full JID for (example omkar@ankur-virtualbox/Spark) along with JID of the receiver (for example ankur@ankur-virtualbox). The receiver then responds with full JID so that they both can exchange messages efficiently. The transitions between the chat states at one of the entities is as shown in the Figure.

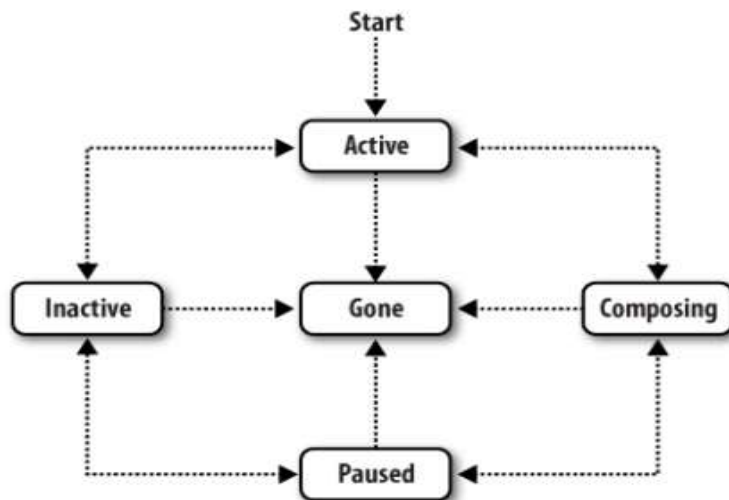


Figure: XMPP Chat states

XMPP <message/> tag can contain variety of bodies like mobile data, notifications from cloud services, alerts errors. It consists of sender id receiver id, type of message. The basic format is as following,

```

<message from='sender' id='sender_id' to='receiver' type='message_type'
        xml:lang='en'>
<subject>subject_if_any</subject>
<body>message_body</body>
</message>

```

Below is an example how Google cloud messaging service uses XMPP to send notifications on cellular phone.

```

<message id="">
  <gcm xmlns="google:mobile:data">
    {
      "to":"REGISTRATION_ID", // "to" replaces "registration_ids"
      "notification": {
        "title": "Portugal vs. Denmark",
        "text": "5 to 1"
      },
      "time_to_live":"600"
    }
  </gcm>
</message>

```

As shown above we can see that <message/> contains data which is in JSON format.

Lab Environment setup

Objective:

To create a simple Instant Messaging server and capture XMPP packets using the chat server and following entities:

Server: **Openfire** version (4.0.2)

Client: **Spark** version (2.7.7)

Gateway service: **Gtalk** (Google Hangouts)

In order to perform XMPP Lab, we implemented the below steps to build the lab setup:

Basic Environment Setup:

Step 1:

We downloaded and installed Oracle Virtual box on windows PC.

<https://www.virtualbox.org/wiki/Downloads>

Step 2:

We installed Mint (Linux) on the virtual box with 2GB RAM and 16GB Virtual Disk Space.

<https://www.linuxmint.com/download.php>

Step 3:

A network analyzing tool (Wireshark), was installed and updated on Mint using the below commands:

\$sudo add-apt-repository ppa:pi-rho/security

\$sudo apt-get update

\$sudo apt-get install wireshark

Openfire Setup:

Step 1:

The first check to perform before installing Openfire was to Install Java. We Checked for Installed Java by using the command **\$java -version** and found that java was not installed and so installed it using the command **\$sudo apt-get install default-jre**

Step 2:

Now, we downloaded openfire_4.0.2_all.deb from igniterealtime.org and installed the downloaded package using **\$sudo dpkg -i openfire_4.0.2_all.deb** command.

```
ankur-VirtualBox ~ # wget http://download.igniterealtime.org/openfire/openfire_4.0.2_all.deb
--2016-04-28 20:05:19-- http://download.igniterealtime.org/openfire/openfire_4.0.2_all.deb
Resolving download.igniterealtime.org (download.igniterealtime.org)... 54.230.144.153, 54.230.144.142, 54.230.144.95, ...
Connecting to download.igniterealtime.org (download.igniterealtime.org)|54.230.144.153|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20357614 (19M) [binary/octet-stream]
Saving to: 'openfire_4.0.2_all.deb'

100%[=====]
2016-04-28 20:05:21 (10.1 MB/s) - 'openfire_4.0.2_all.deb' saved [20357614/20357614]
```

Figure: Download Openfire

```
ankur-VirtualBox ~ # dpkg -i openfire_4.0.2_all.deb
Selecting previously unselected package openfire.
(Reading database ... 163763 files and directories currently installed.)
Preparing to unpack openfire_4.0.2_all.deb ...
Unpacking openfire (4.0.2) ...
Setting up openfire (4.0.2) ...
```

Figure: Install Openfire

Step 3:

Now, as we have installed the openfire server on Linux Mint, the openfire service can be stopped and started from command prompt by using the commands **\$sudo /etc/init.d/openfire stop** and **\$sudo /etc/init.d/openfire start** respectively.

Step 4:

Openfire server is now started on the localhost of the Linux mint machine. We accessed the server by browsing <http://localhost:9090> from the browser and started with the server settings:

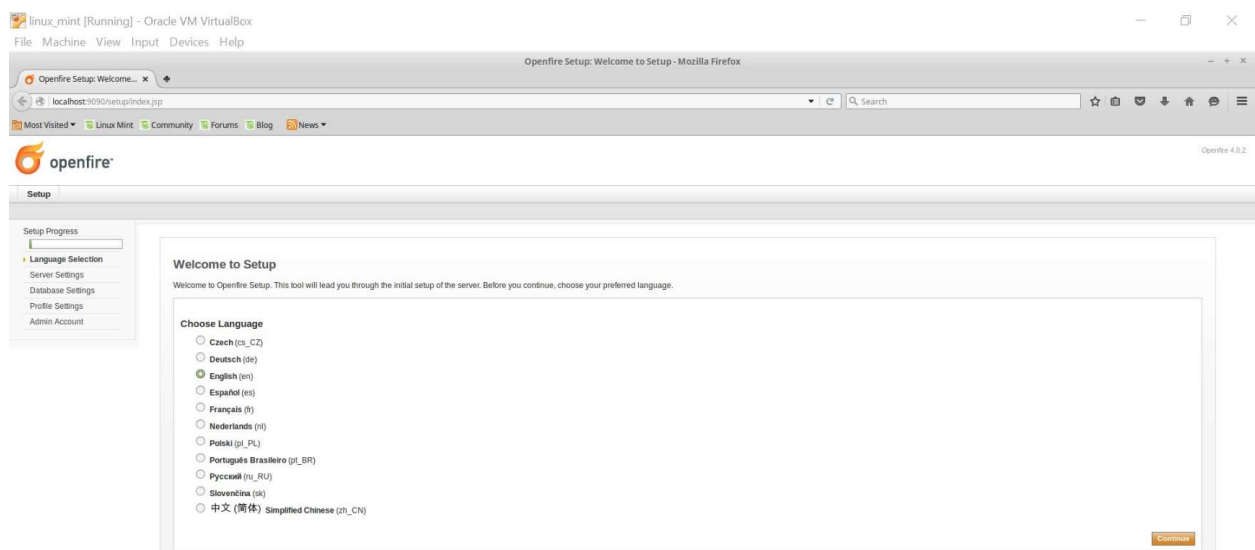


Figure: Server Language selection

In server settings, domain should be the local host name or the ip address of machine as shown below:

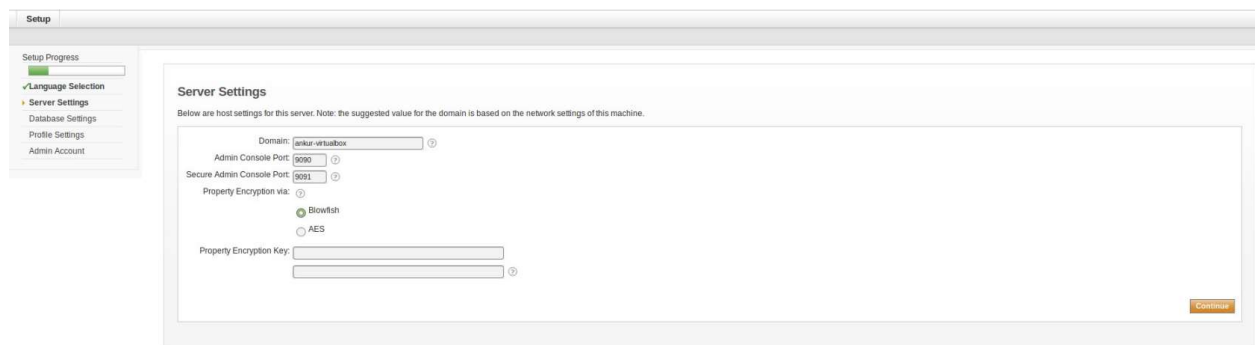


Figure: Server settings

Since we had planned to capture the XMPP packets for the protocol analysis by simple deployment, we used Embedded database.

The screenshot shows the 'Database Settings' step of the Openfire setup wizard. On the left, a 'Setup Progress' sidebar lists: Setup Progress (100%), Language Selection (checked), Server Settings (checked), Database Settings (checked), Profile Settings (checked), and Admin Account (checked). The main content area is titled 'Database Settings' and instructs the user to 'Choose how you would like to connect to the Openfire database.' There are two radio button options: 'Standard Database Connection' (unselected) and 'Embedded Database' (selected). The 'Embedded Database' option is described as using an embedded database powered by HSQLDB, requiring no external configuration, and is noted as an easy way to get up and running quickly, though it may not offer the same performance as an external database. A 'Continue' button is located at the bottom right.

Figure: Database settings

Since we had planned to capture the XMPP packets for the protocol analysis by simple deployment, we used default profile settings.

The screenshot shows the 'Profile Settings' step of the Openfire setup wizard. The 'Setup Progress' sidebar is identical to the previous screen. The main content area is titled 'Profile Settings' and instructs the user to 'Choose the user and group system to use with the server.' There are three radio button options: 'Default' (selected), 'Only Hashed Passwords' (unselected), and 'Directory Server (LDAP)' (unselected). The 'Default' option is described as storing users and groups in the server database, being the best for simple deployments. The 'Only Hashed Passwords' option is described as storing only non-reversible password hashes, supporting PLAIN and SCRAM-SHA-1. The 'Directory Server (LDAP)' option is described as integrating with a directory server like Active Directory or OpenLDAP. A 'Continue' button is at the bottom right.

Figure: Profile settings

Here, the administrator account details are entered, the username and password are used to login to the openfire console.

The screenshot shows the 'Administrator Account' step of the Openfire setup wizard. The 'Setup Progress' sidebar is identical. The main content area is titled 'Administrator Account' and instructs the user to 'Enter settings for the system administrator account (username of "admin") below. It is important to choose a password for the account that cannot be easily guessed -- for example, at least eight characters long and containing a mix of letters and numbers. You can skip this step if you have already setup your admin account. If you skip this step during the new installation, the default password would be "admin".' There are three input fields: 'Admin Email Address' (containing 'sauris1@gmail.com'), 'New Password' (masked with asterisks), and 'Confirm Password' (masked with asterisks). A 'Skip This Step' button is disabled, and a 'Continue' button is active. A small note below the email field says 'A valid email address for the admin account.'

Figure: Administrator Account settings

The screenshot shows the 'Setup Complete' screen of the Openfire setup wizard. The 'Setup Progress' sidebar is identical. The main content area is titled 'Setup Complete!' and states 'This installation of Openfire is now complete. To continue:'. There is a single 'Login to the admin console' button.

Figure: Openfire Setup Complete

After the initial setup completion, we logged into the Openfire console using the administrator credentials



Figure: Openfire Console

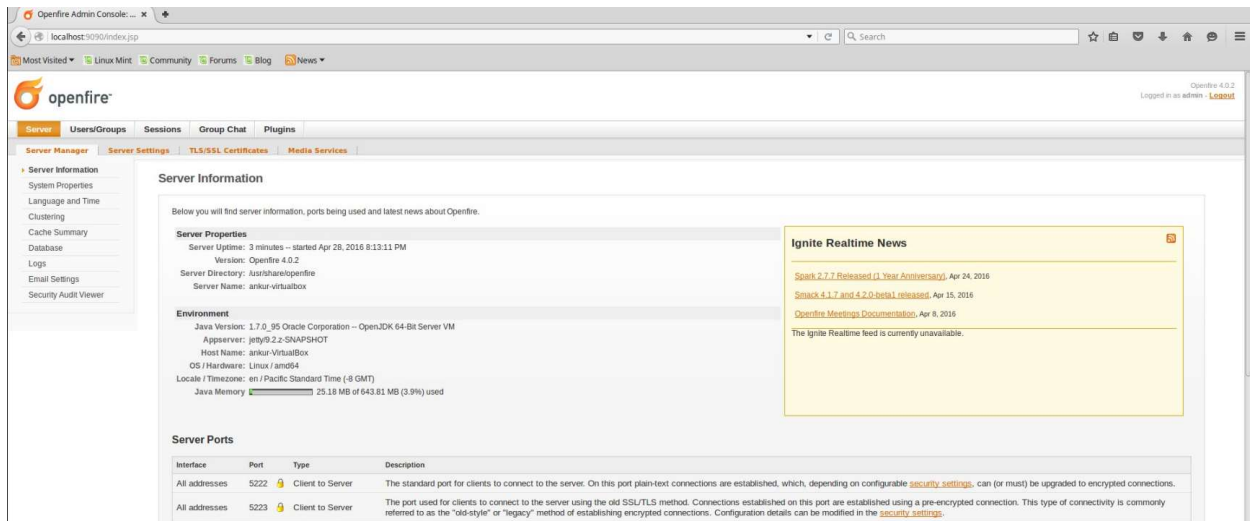


Figure: Server Information

By using the above setup, we implemented the simple XMPP architecture and captured the required packets using the loopback interface and Wireshark.

Step 5:

Now, in order to access public instant messaging networks such as GTalk, IRC, Facebook messenger, AOL messenger, etc., and to explore the complex XMPP architecture, we installed an additional plugin in Openfire, namely Kraken IM Gateway which is used to provide connectivity.

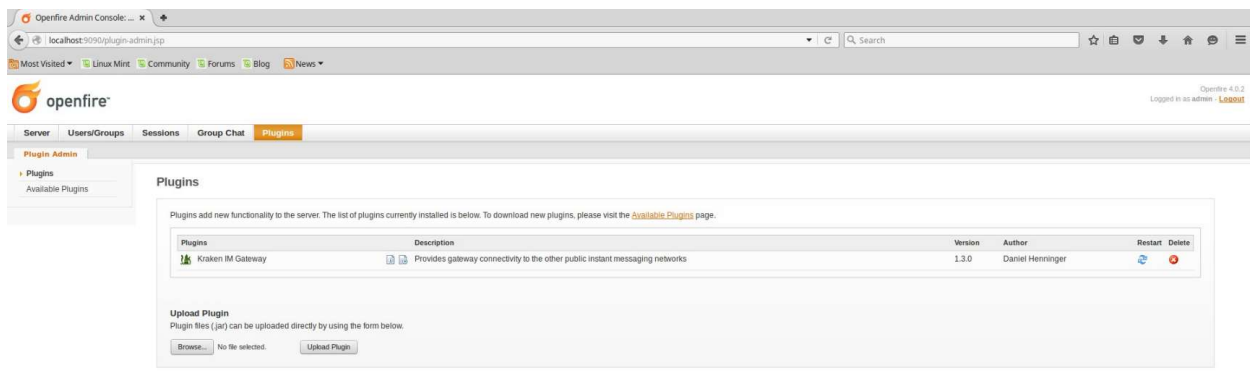


Figure: Installed Plugins (Kraken IM Gateway)

Step 6:

Now, we selected the Gateway Settings by going to Server Tab and then selected Gtalk (Google Hangouts) gateway service and tried the connection to the corresponding server.

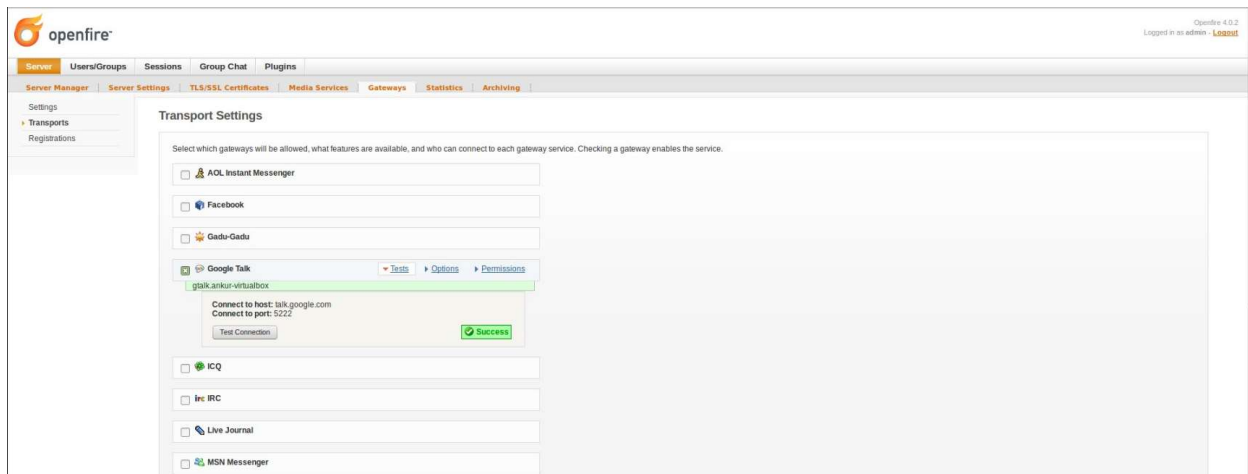


Figure: Gateway service selection (Gtalk)

Step 7:

Now, we added a new registration for Gtalk gateway service using the username and password which can be used to login via Spark Client.

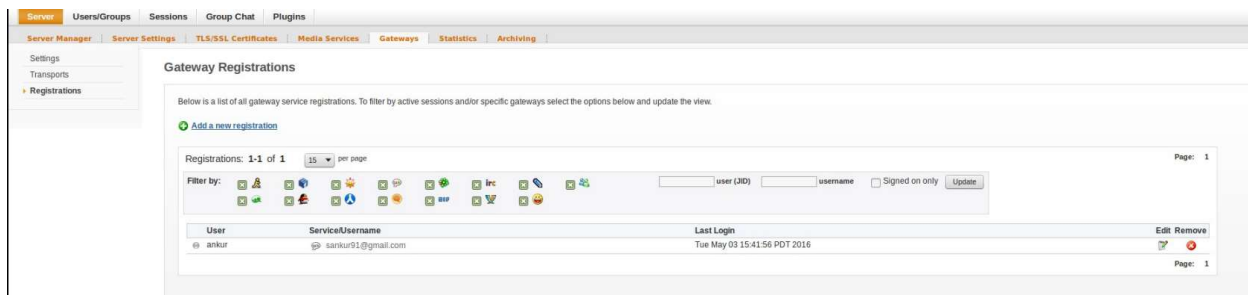


Figure: Gateway registration

Note: For our deployment we selected Gtalk (Google Hangouts) but one can select any gateway service and check the connection to the corresponding server.

Spark Setup:

Step 1:

Now, to Install Spark IM Client, we download the package spark_2_7_7.tar.gz from igniterealtime.org in in /opt path and unzipped the package using `$ tar -zxvf spark_2_7_7.tar.gz`

```

ankur-VirtualBox opt # wget http://download.igniterealtime.org/spark/spark_2_7_7.tar.gz
--2016-04-28 20:34:48-- http://download.igniterealtime.org/spark/spark_2_7_7.tar.gz
Resolving download.igniterealtime.org (download.igniterealtime.org)... 54.230.144.153, 54.230.144.25, 54.230.144.196, ...
Connecting to download.igniterealtime.org (download.igniterealtime.org)[54.230.144.153]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 119170564 (114M) [application/x-tar]
Saving to: 'spark_2_7_7.tar.gz'

100%[=====] 119170564 7.64 MB/s
2016-04-28 20:35:03 (7.64 MB/s) - 'spark_2_7_7.tar.gz' saved [119170564/119170564]

ankur-VirtualBox opt #

```

Figure: Download and Install spark

Step 2:

Now, as the spark IM client was installed, we created a desktop entry by which the Client can be accessed by the user in friendly manner by creating a file /usr/share/applications/spark.desktop

```

GNU nano 2.2.6 File: /usr/share/applications/spark.desktop

[Desktop Entry]
Name=Spark
Version=2.7.7
GenericName=Spark
X-GNOME-FullName=Spark
Comment=ignite realtime Spark IM Client
Type=Application
Categories=Application;Utility;
Path=/opt/spark
Exec=/bin/bash Spark
Terminal=false
StartupNotify=true
Icon=/opt/spark/spark.png
TargetEnvironment=Unity

```

Figure: Desktop Entry Creation using Nano editor

Step 3:

To get the Icon for the desktop entry we downloaded the Spark Icon (spark.png) in /opt/Spark path as shown:

```

ankur@ankur-VirtualBox /opt/Spark $ sudo wget https://dl.dropbox.com/u/50880014/spark.png
--2016-04-28 22:31:58-- https://dl.dropbox.com/u/50880014/spark.png
Resolving dl.dropbox.com (dl.dropbox.com)... 108.160.172.70
Connecting to dl.dropbox.com (dl.dropbox.com)[108.160.172.70]:443... connected.
HTTP request sent, awaiting response... 302 FOUND
Location: https://dl.dropboxusercontent.com/u/50880014/spark.png [following]
--2016-04-28 22:31:58-- https://dl.dropboxusercontent.com/u/50880014/spark.png
Resolving dl.dropboxusercontent.com (dl.dropboxusercontent.com)... 199.47.217.69
Connecting to dl.dropboxusercontent.com (dl.dropboxusercontent.com)[199.47.217.69]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9575 (9.4K) [image/png]
Saving to: 'spark.png'

100%[=====] 9,575 --.-K/s in 0s
2016-04-28 22:31:58 (245 MB/s) - 'spark.png' saved [9575/9575]

```

The spark IM Client installation was now complete and we Launched it using the Spark icon on desktop and logged in using the credentials given during Openfire setup. The server here was entered as localhost name or IP address, since it is the name of the Openfire server.

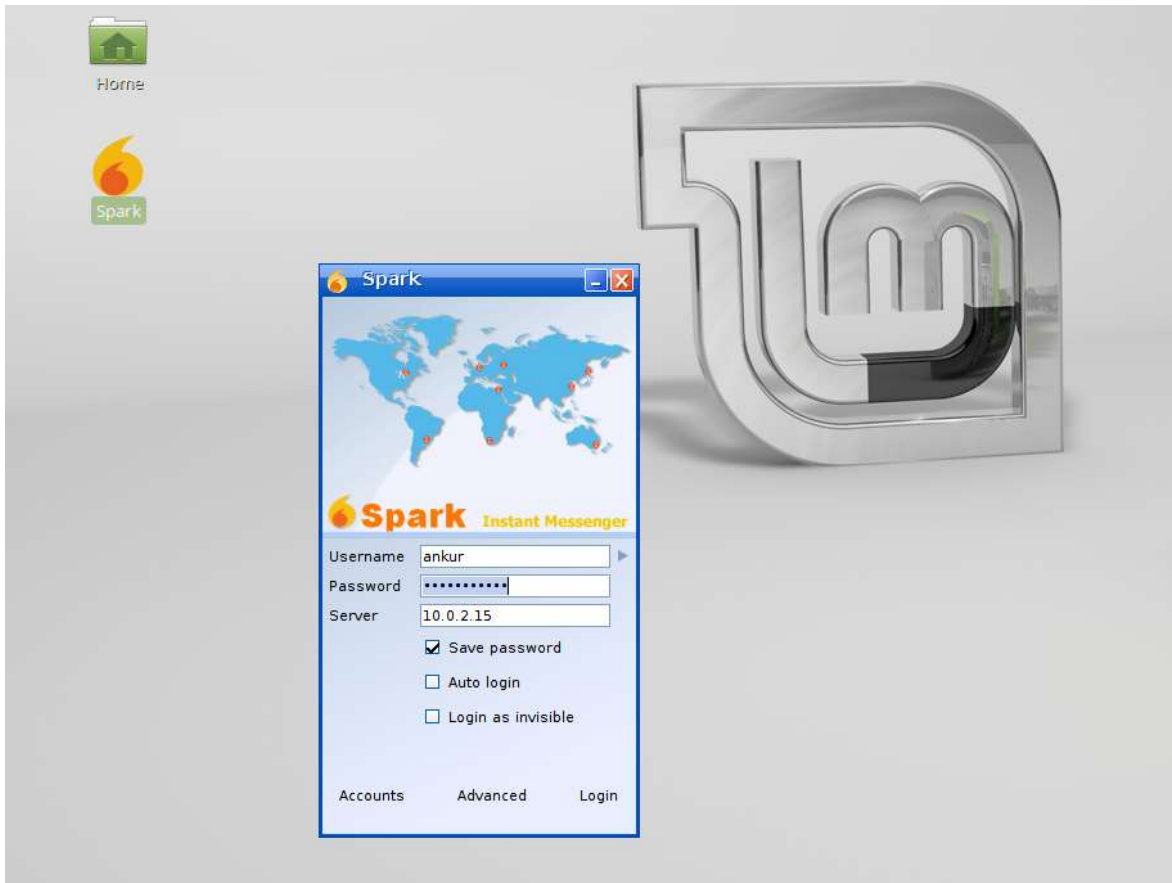


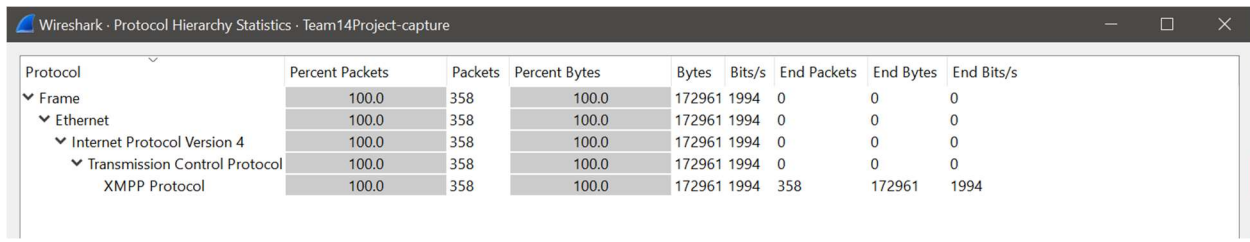
Figure: Spark Login using Desktop Entry

Procedure:

We captured the XMPP packets for the both the architectures using Wireshark, a network analyzing tool.

- Firstly, we opened Wireshark and kept it running in background to capture the required XMPP packets during Chatting between the two entities.
- By logging into the spark IM client we exchanged the chat messages between two entities and also changed the presence status on the same installed Openfire server.
- Now, we entered the Gmail credentials and tried to communicate using Gtalk service between the two entities, as we had already installed the kraken Plugin along with the Gtalk gateway service in Openfire.
- The captured packets were filtered using the XMPP filter to study and analyze further.
- Various captured XMPP packets and their explanation is described below.

Protocol Hierarchy:



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	358	100.0	172961	1994	0	0	0
▼ Ethernet	100.0	358	100.0	172961	1994	0	0	0
▼ Internet Protocol Version 4	100.0	358	100.0	172961	1994	0	0	0
▼ Transmission Control Protocol	100.0	358	100.0	172961	1994	0	0	0
XMPP Protocol	100.0	358	100.0	172961	1994	358	172961	1994

Figure: Protocol Hierarchy as observed from Wireshark

Flow Graph:

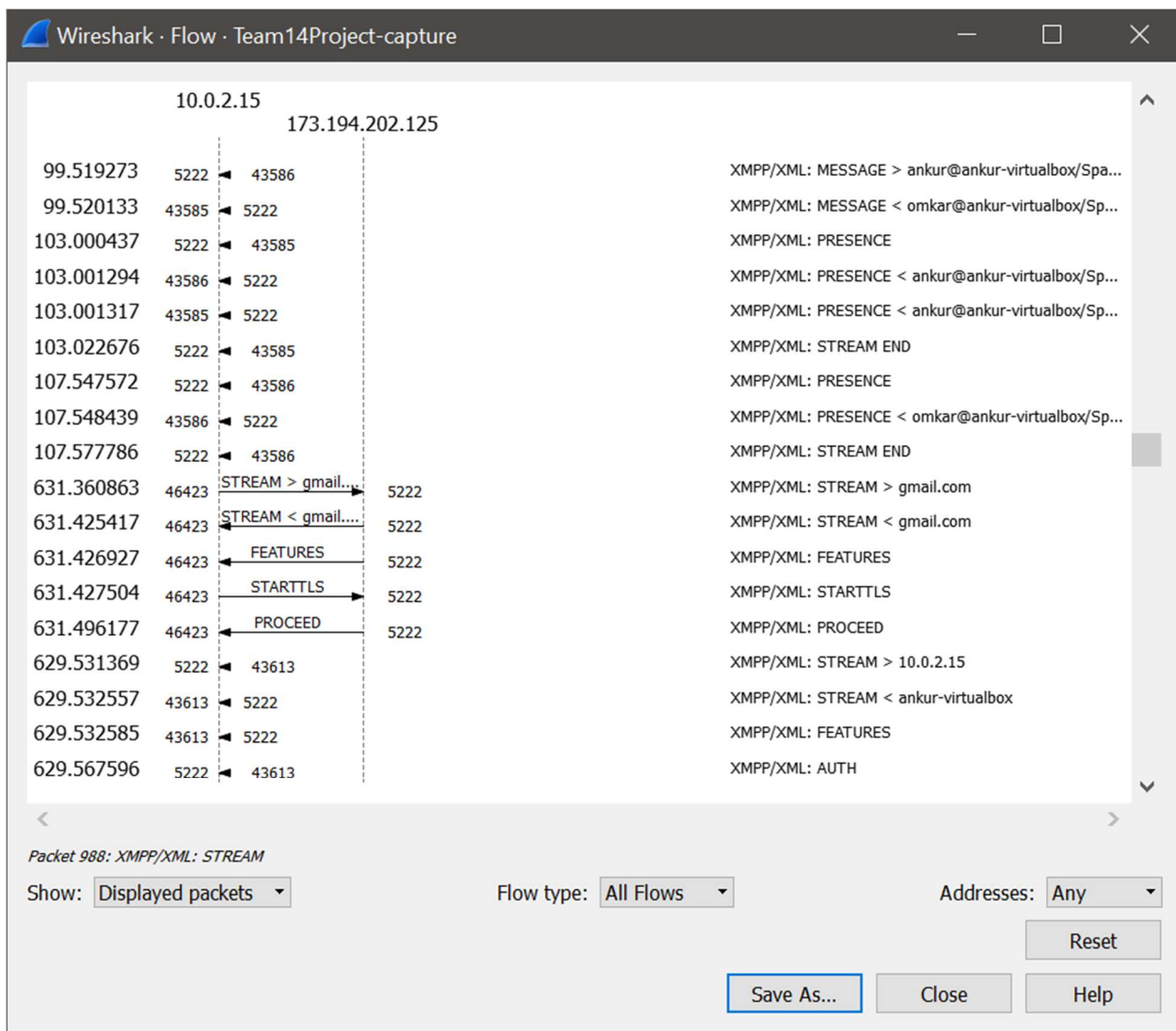


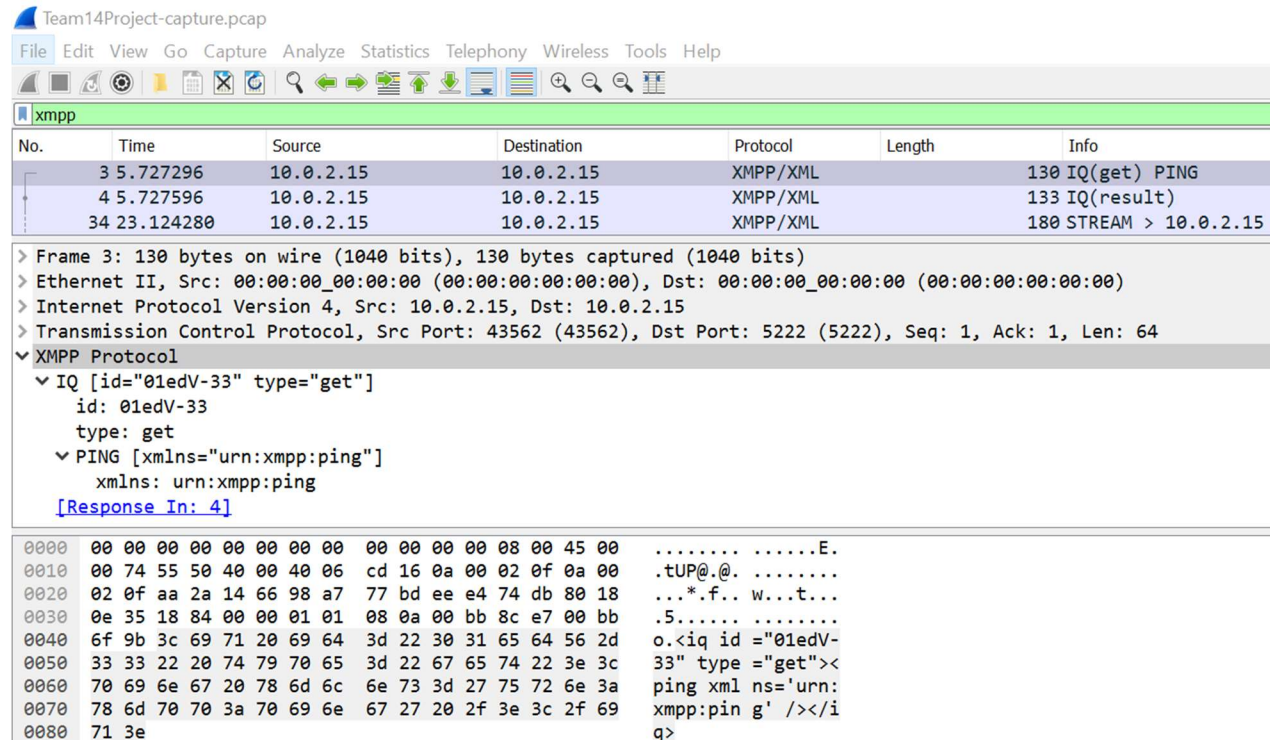
Figure: Flow graph captured from Wireshark.

Flow graph explains about the various flow events that take place during the execution of XMPP lab.

Captured XMPP packets

Ping Packets

As protocol states, Client sends an IQ-get containing a <ping/> element qualified by the 'urn:xmpp:ping' namespace. The pinged entity returns either an IQ-result (if it supports the namespace) or an IQ-error (if it does not). In our case it has returned IQ-result as shown below:



The image shows a Wireshark capture of XMPP packets. The packet list pane displays three packets: a ping request (Frame 3), a ping result (Frame 4), and a stream packet (Frame 34). The packet details pane for Frame 3 shows the XMPP protocol structure, including the IQ-get and PING elements. The packet bytes pane shows the raw data of the ping request.

No.	Time	Source	Destination	Protocol	Length	Info
3	5.727296	10.0.2.15	10.0.2.15	XMPP/XML	130	IQ(get) PING
4	5.727596	10.0.2.15	10.0.2.15	XMPP/XML	133	IQ(result)
34	23.124280	10.0.2.15	10.0.2.15	XMPP/XML	180	STREAM > 10.0.2.15

Frame 3: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits)

Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15

Transmission Control Protocol, Src Port: 43562 (43562), Dst Port: 5222 (5222), Seq: 1, Ack: 1, Len: 64

XMPP Protocol

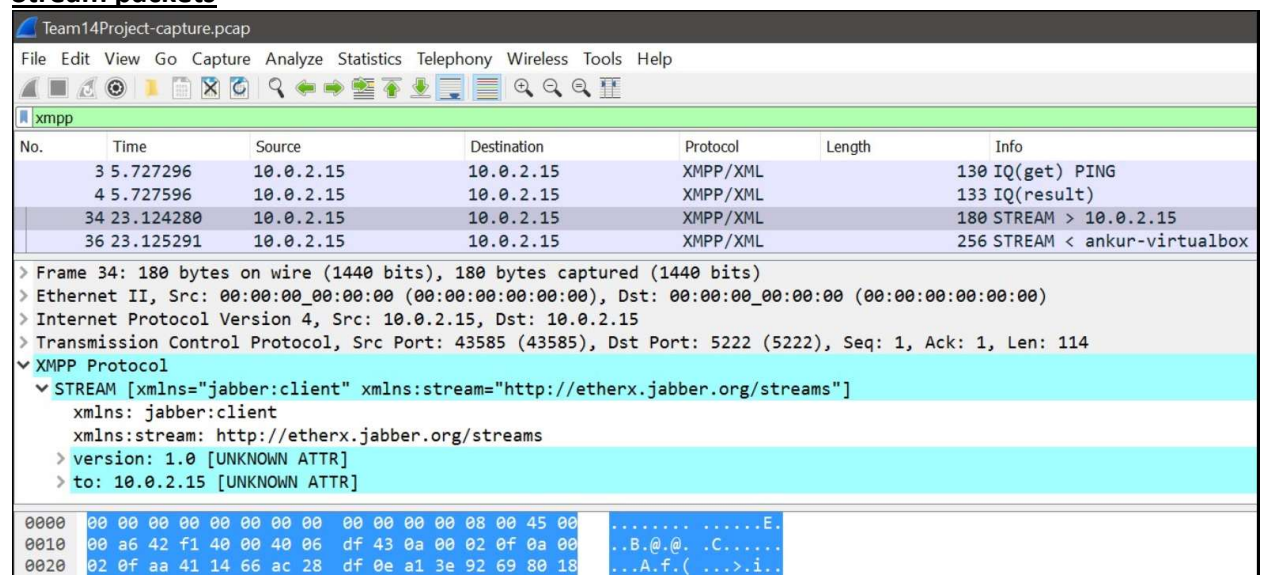
- IQ [id="01edV-33" type="get"]
 - id: 01edV-33
 - type: get
 - PING [xmlns="urn:xmpp:ping"]
 - xmlns: urn:xmpp:ping

[Response In: 4]

0000 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.
0010 00 74 55 50 40 00 40 06 cd 16 0a 00 02 0f 0a 00 .tUP@.@.
0020 02 0f aa 2a 14 66 98 a7 77 bd ee e4 74 db 80 18 ...*.f.. w...t...
0030 0e 35 18 84 00 00 01 01 08 0a 00 bb 8c e7 00 bb .5.....
0040 6f 9b 3c 69 71 20 69 64 3d 22 30 31 65 64 56 2d o.<iq id = "01edV-
0050 33 33 22 20 74 79 70 65 3d 22 67 65 74 22 3e 3c 33" type = "get"><
0060 70 69 6e 67 20 78 6d 6c 6e 73 3d 27 75 72 6e 3a ping xml ns='urn:
0070 78 6d 70 70 3a 70 69 6e 67 27 20 2f 3e 3c 2f 69 xmpp:pin g' /></i
0080 71 3e q>

Figure: XMPP Ping packets

Stream packets



The image shows a Wireshark capture of XMPP packets. The packet list pane displays four packets: a ping request (Frame 3), a ping result (Frame 4), a stream packet (Frame 34), and another stream packet (Frame 36). The packet details pane for Frame 34 shows the XMPP protocol structure, including the STREAM element. The packet bytes pane shows the raw data of the stream packet.

No.	Time	Source	Destination	Protocol	Length	Info
3	5.727296	10.0.2.15	10.0.2.15	XMPP/XML	130	IQ(get) PING
4	5.727596	10.0.2.15	10.0.2.15	XMPP/XML	133	IQ(result)
34	23.124280	10.0.2.15	10.0.2.15	XMPP/XML	180	STREAM > 10.0.2.15
36	23.125291	10.0.2.15	10.0.2.15	XMPP/XML	256	STREAM < ankur-virtualbox

Frame 34: 180 bytes on wire (1440 bits), 180 bytes captured (1440 bits)

Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15

Transmission Control Protocol, Src Port: 43585 (43585), Dst Port: 5222 (5222), Seq: 1, Ack: 1, Len: 114

XMPP Protocol

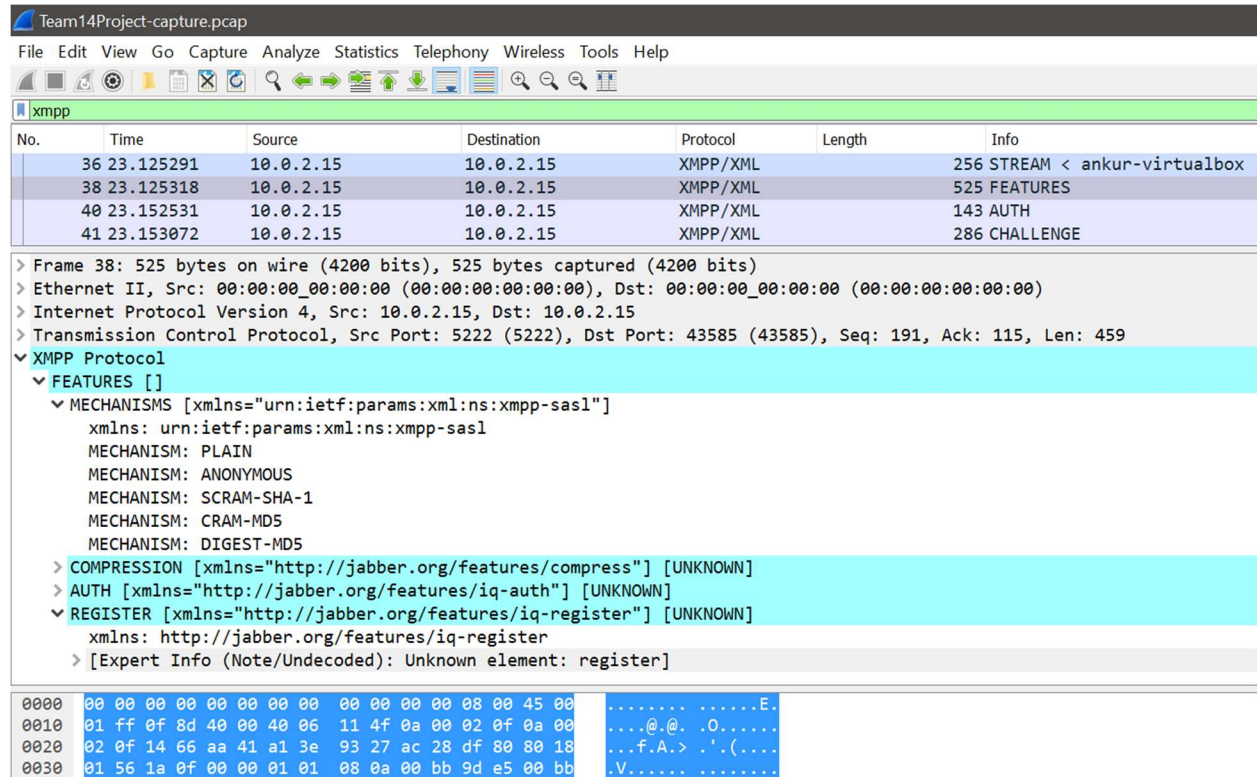
- STREAM [xmlns="jabber:client" xmlns:stream="http://etherx.jabber.org/streams"]
 - xmlns: jabber:client
 - xmlns:stream: http://etherx.jabber.org/streams
 - version: 1.0 [UNKNOWN ATTR]
 - to: 10.0.2.15 [UNKNOWN ATTR]

0000 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.
0010 00 a6 42 f1 40 00 40 06 df 43 0a 00 02 0f 0a 00 ..B.@.@. .C.....
0020 02 0f aa 41 14 66 ac 28 df 0e a1 3e 92 69 80 18 ...A.f.(...>.i..

Figure: XMPP Stream packets

Streams are nothing but the containers for the XML messages passed from both the ways that is from client to server and vice versa. Once the stream is opened from both the ends client as well as server can send XML messages to each other without any interruption. Later client as well as server will send stanzas to each other over the two streams opened.

Features packet



The image shows a Wireshark capture of an XMPP stream. The packet list shows four packets: a STREAM packet (256 bytes), a FEATURES packet (525 bytes), an AUTH packet (143 bytes), and a CHALLENGE packet (286 bytes). The packet details pane for the FEATURES packet (Frame 38) is expanded, showing the following structure:

- MECHANISMS [xmlns="urn:ietf:params:xml:ns:xmpp-sasl"]
 - xmlns: urn:ietf:params:xml:ns:xmpp-sasl
 - MECHANISM: PLAIN
 - MECHANISM: ANONYMOUS
 - MECHANISM: SCRAM-SHA-1
 - MECHANISM: CRAM-MD5
 - MECHANISM: DIGEST-MD5
- COMPRESSION [xmlns="http://jabber.org/features/compress"] [UNKNOWN]
- AUTH [xmlns="http://jabber.org/features/iq-auth"] [UNKNOWN]
- REGISTER [xmlns="http://jabber.org/features/iq-register"] [UNKNOWN]
 - xmlns: http://jabber.org/features/iq-register

The packet bytes pane at the bottom shows the raw data in hexadecimal and ASCII.

Figure: XMPP Features packet

As server has proactively started a stream from Server to Client as well it sends the Stream Features. We can see the features of the stream in Features packets. It specifies SASL mechanisms available for client to choose like DIGEST-MD5, PLAIN etc. under <mechanism/> tag. It also provides tag like <compression/> <AUTH/> and <REGISTER/>

Authentication(Auth) packet

This is sent from client to server in response to the Feature packet and to notify which authentication it has selected mentioned in Feature.

No.	Time	Source	Destination	Protocol	Length	Info
36	23.125291	10.0.2.15	10.0.2.15	XMPP/XML	256	STREAM < ankur-virtualbox
38	23.125318	10.0.2.15	10.0.2.15	XMPP/XML	525	FEATURES
40	23.152531	10.0.2.15	10.0.2.15	XMPP/XML	143	AUTH
41	23.153072	10.0.2.15	10.0.2.15	XMPP/XML	286	CHALLENGE
42	23.154972	10.0.2.15	10.0.2.15	XMPP/XML	504	RESPONSE

> Frame 40: 143 bytes on wire (1144 bits), 143 bytes captured (1144 bits)
 > Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 > Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15
 > Transmission Control Protocol, Src Port: 43585 (43585), Dst Port: 5222 (5222), Seq: 115, Ack: 650, Len: 77
 > XMPP Protocol
 > AUTH [xmlns="urn:ietf:params:xml:ns:xmpp-sasl" mechanism="DIGEST-MD5"]
 xmlns: urn:ietf:params:xml:ns:xmpp-sasl
 mechanism: DIGEST-MD5
 CDATA: (empty)

0000 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.
 0010 00 81 42 f4 40 00 40 06 df 65 0a 00 02 0f 0a 00 ...B.@. .e.....
 0020 02 0f aa 41 14 66 ac 28 df 80 a1 3e 94 f2 80 18 ...A.f.(...>...

Figure: XMPP Authentication (Auth) packet

Success Packets

If the server accepts the authorization, it sends back a stream with “success” tag. <success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/> or: Server: If the password does not match the user name, or there is an error on encoding, the server will sends a failure stream. <failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>

No.	Time	Source	Destination	Protocol	Length	Info
36	23.125291	10.0.2.15	10.0.2.15	XMPP/XML	256	STREAM < ankur-virtualbox
38	23.125318	10.0.2.15	10.0.2.15	XMPP/XML	525	FEATURES
40	23.152531	10.0.2.15	10.0.2.15	XMPP/XML	143	AUTH
41	23.153072	10.0.2.15	10.0.2.15	XMPP/XML	286	CHALLENGE
42	23.154972	10.0.2.15	10.0.2.15	XMPP/XML	504	RESPONSE
43	23.157293	10.0.2.15	10.0.2.15	XMPP/XML	182	SUCCESS
44	23.159965	10.0.2.15	10.0.2.15	XMPP/XML	187	STREAM > ankur-virtualbox
45	23.160572	10.0.2.15	10.0.2.15	XMPP/XML	559	STREAM < ankur-virtualbox
46	23.160833	10.0.2.15	10.0.2.15	XMPP/XML	170	STREAM < ankur-virtualbox

> Frame 43: 182 bytes on wire (1456 bits), 182 bytes captured (1456 bits)
 > Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 > Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15
 > Transmission Control Protocol, Src Port: 5222 (5222), Dst Port: 43585 (43585), Seq: 870, Ack: 630, Len: 116
 > XMPP Protocol
 > SUCCESS [xmlns="urn:ietf:params:xml:ns:xmpp-sasl"]
 > [Expert Info (Chat/Response): SUCCESS]
 xmlns: urn:ietf:params:xml:ns:xmpp-sasl

0000 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.
 0010 00 a8 0f 8f 40 00 40 06 12 a4 0a 00 02 0f 0a 00 ...@.@.
 0020 02 0f 14 66 aa 41 a1 3e 95 ce ac 28 e1 83 80 18 ...f.A.> ...(<....

Figure: XMPP Success packet

IQ packets

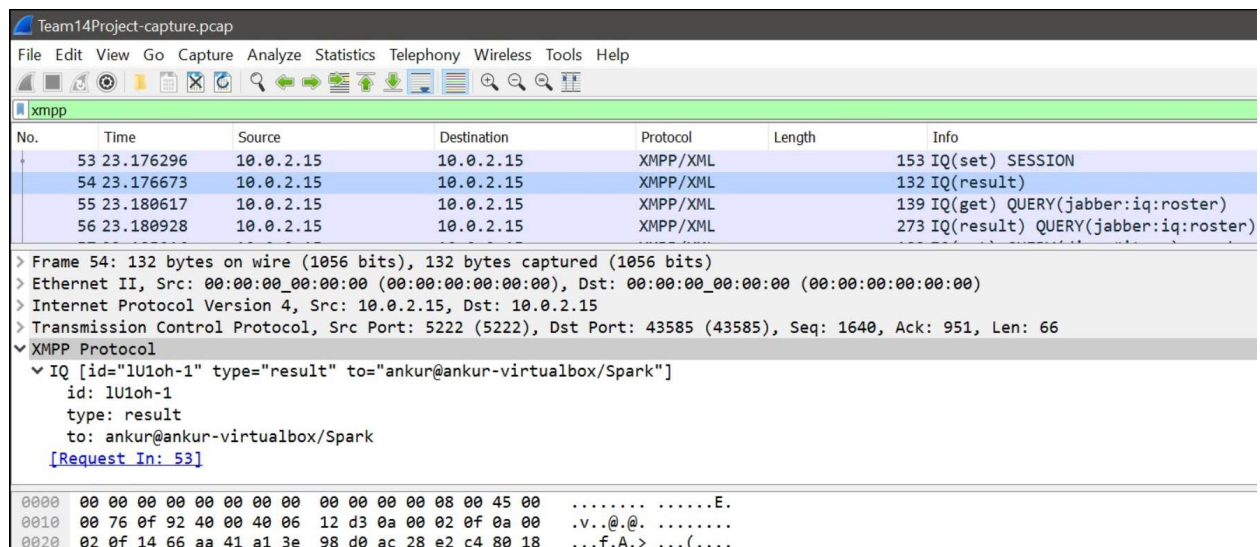


Figure: XMPP IQ packet

1.Bind (Type=set)

Client ask a server to bind a resource or itself binds a resource. In our case client itself has bound the resource with name 'Spark' which will be appended to the JID to give us full JID

2.Result (Type=result)

Server returns full JID to client in response to this in our case it is ankur@ankur-virtualbox/Spark

3. Session (Type=set)

As client binds a resource with server it requests new session to the server.

4. Session (Type=result)

Server sends response specifying whether session is established or not.

5. Query (Type=get)

Now the client in our case omkar@ankur-virtualbox/Spark can query the roster for the other users to have a chat which can be seen in packet

6. Query (Type=response)

Server responds with the roster entries with JID of the user/users. In our case it contains details of the user named ankur with JID ankur@ankur-virtualbox

Presence Packets

The client (with JID omkar@ankur-virtualbox/Spark) sends the presence packet to the server and informs server about his status

Server then sends this Presence packet to ankur@ankur-virtualbox

The user2(with JID ankur@ankur-virtualbox/Spark) then sends its status to user with JID omkar@ankur-virtualbox/Spark

These two clients then exchange PRESENCE packets with their status.

No.	Time	Source	Destination	Protocol	Length	Info
149	40.330153	10.0.2.15	10.0.2.15	XMPP/XML		163 IQ(get) VCARD
150	40.330652	10.0.2.15	10.0.2.15	XMPP/XML		163 IQ(result) VCARD
155	41.132730	10.0.2.15	10.0.2.15	XMPP/XML		145 PRESENCE
156	41.133420	10.0.2.15	10.0.2.15	XMPP/XML		209 PRESENCE < omkar@ankur-virtualbox/Spark
157	41.133713	10.0.2.15	10.0.2.15	XMPP/XML		215 PRESENCE < ankur@ankur-virtualbox/Spark

> Frame 156: 209 bytes on wire (1672 bits), 209 bytes captured (1672 bits)

> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)

> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.15

> Transmission Control Protocol, Src Port: 5222 (5222), Dst Port: 43585 (43585), Seq: 16241, Ack: 3616, Len: 143

▼ XMPP Protocol

▼ PRESENCE [id="Qejg0-6"]

from: omkar@ankur-virtualbox/Spark

id: Qejg0-6

to: ankur@ankur-virtualbox

priority: 1

▼ STATUS [value="Online"]

value: Online

0000 00 00 00 00 00 00 00 00 00 00 08 00 00 45 00E.

0010 00 c3 0f ab 40 00 00 06 12 6d 0a 00 02 0f 0a 00 ...@.@. .m.....

0020 02 0f 14 66 aa 41 a1 3e d1 d9 ac 28 ed 2d 80 18 ...f.A.> ...(-..

0030 01 77 18 d3 00 00 01 01 08 0a 00 bb af 7b 00 bb .W.....{..

Figure: XMPP Presence packet

Message Packets

Message packets contains actual text messages to be conveyed to the clients. We can see actual text messages sent in Body of the message.

Message can contain any tag for example in case of cellular notifications it contains notification body in json format.

No.	Time	Source	Destination	Protocol	Length	Info
206	54.034094	10.0.2.15	10.0.2.15	XMPP/XML		179 IQ(get) QUERY
207	54.034543	10.0.2.15	10.0.2.15	XMPP/XML		216 IQ(result) QUERY
209	62.929367	10.0.2.15	10.0.2.15	XMPP/XML		265 MESSAGE > omkar@ankur-virtualbox
210	62.930032	10.0.2.15	10.0.2.15	XMPP/XML		265 MESSAGE < ankur@ankur-virtualbox/Spark

> Transmission Control Protocol, Src Port: 43585 (43585), Dst Port: 5222 (5222), Seq: 3616, Ack: 16384, Len: 199

▼ XMPP Protocol

▼ MESSAGE [id="lU1oh-25" type="chat"]

from: ankur@ankur-virtualbox/Spark

id: lU1oh-25

to: omkar@ankur-virtualbox

type: chat

▼ THREAD [value="78bmF4"]

value: 78bmF4

▼ BODY [value="Hi"]

value: Hi

▼ X EVENT [condition="offline/composing"]

xmlns: jabber:x:event

CONDITION: offline/composing

0000 00 00 00 00 00 00 00 00 00 00 08 00 00 45 00E.

0010 00 fb 43 21 40 00 00 06 de be 0a 00 02 0f 0a 00 ...C!@.@.

0020 02 0f aa 41 14 66 ac 28 ed 2d a1 3e d2 68 80 18 ...A.f.(.->.h..

Figure: XMPP Message packet

DNS Lookup Query/response packets

As we have used Gtalk as a transport while capturing packets in a scenario where XMPP client on our local machine chats with the Gtalk user using Kraken gateway. We have captured DNS packets as shown here where our local machine sends DNS queries to DNS server at 10.10.90.50 and got the google servers IP in response which is 173.194.202.125.

The image shows a Wireshark capture of DNS traffic. The packet list shows three DNS packets: a query (No. 356), a response (No. 357), and another response (No. 358). The details pane for packet 357 shows a standard query response for 'talk.google.com' with a CNAME record pointing to 'talk.l.google.com' and an A record for '173.194.202.125'.

No.	Time	Source	Destination	Protocol	Length	Info
356	631.324377	10.0.2.15	10.10.90.50	DNS	75	Standard query 0x5d99 AAAA talk.google.com
357	631.328595	10.10.90.50	10.0.2.15	DNS	112	Standard query response 0xd95d A talk.google.com CNAME talk.l.google.com A...
358	631.330619	10.10.90.50	10.0.2.15	DNS	124	Standard query response 0x5d99 AAAA talk.google.com CNAME talk.l.google.co...

Frame 357: 112 bytes on wire (896 bits), 112 bytes captured (896 bits)
 Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: CadmusCo_49:a5:48 (08:00:27:49:a5:48)
 Internet Protocol Version 4, Src: 10.10.90.50, Dst: 10.0.2.15
 User Datagram Protocol, Src Port: 53 (53), Dst Port: 20747 (20747)
 Domain Name System (response)
 [Request In: 355]
 [Time: 0.004356000 seconds]
 Transaction ID: 0xd95d
 Flags: 0x8100 Standard query response, No error
 Questions: 1
 Answer RRs: 2
 Authority RRs: 0
 Additional RRs: 0
 Queries
 talk.google.com: type A, class IN
 Name: talk.google.com
 [Name Length: 15]
 [Label Count: 3]
 Type: A (Host Address) (1)
 Class: IN (0x0001)
 Answers
 talk.google.com: type CNAME, class IN, cname talk.l.google.com
 talk.l.google.com: type A, class IN, addr 173.194.202.125

Figure: DNS Lookup Query

SSL/TLS Secure Connection:

As we can see STARTTLS message has been sent to google server before proceeding to SASL negotiations.

The image shows a Wireshark capture of XMPP traffic. The packet list shows five packets: two TCP ACKs (Nos. 367, 369) and three XMPP/XML packets (Nos. 368, 370, 371). The details pane for packet 368 shows a STARTTLS message with the XML namespace 'urn:ietf:params:xml:ns:xmpp-tls'.

No.	Time	Source	Destination	Protocol	Length	Info
367	631.427016	10.0.2.15	173.194.202.125	TCP	54	46423 → 5222 [ACK]
368	631.427504	10.0.2.15	173.194.202.125	XMPP/XML	105	STARTTLS
369	631.427930	173.194.202.125	10.0.2.15	TCP	60	5222 → 46423 [ACK]
370	631.496177	173.194.202.125	10.0.2.15	XMPP/XML	104	PROCEED
371	631.506772	10.0.2.15	173.194.202.125	TLSv1.2	276	Client Hello

Frame 368: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)
 Ethernet II, Src: CadmusCo_49:a5:48 (08:00:27:49:a5:48), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
 Internet Protocol Version 4, Src: 10.0.2.15, Dst: 173.194.202.125
 Transmission Control Protocol, Src Port: 46423 (46423), Dst Port: 5222 (5222), Seq: 115, Ack: 380, Len: 51
 XMPP Protocol
 STARTTLS [xmlns="urn:ietf:params:xml:ns:xmpp-tls"]
 xmlns: urn:ietf:params:xml:ns:xmpp-tls

Figure: STARTTLS Packet

As described in the report, XMPP uses SSL/TLS protocols for channel encryption and security. Following are the TLS packets captured during the exchange of XMPP packets in the chat session.

Team14Project-capture.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
382	631.549506	173.194.202.125	10.0.2.15	TCP		60 5222 → 46423 [ACK] Seq=3962 Ack=514 Win=65535 Len=0
383	631.647614	173.194.202.125	10.0.2.15	TLSv1.2		105 Change Cipher Spec, Hello Request, Hello Request
384	631.647936	10.0.2.15	173.194.202.125	TLSv1.2		197 Application Data
385	631.648269	173.194.202.125	10.0.2.15	TCP		60 5222 → 46423 [ACK] Seq=4013 Ack=657 Win=65535 Len=0
386	631.712324	173.194.202.125	10.0.2.15	TLSv1.2		447 Application Data, Application Data

> Frame 383: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)

> Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: CadmusCo_49:a5:48 (08:00:27:49:a5:48)

> Internet Protocol Version 4, Src: 173.194.202.125, Dst: 10.0.2.15

> Transmission Control Protocol, Src Port: 5222 (5222), Dst Port: 46423 (46423), Seq: 3962, Ack: 514, Len: 51

Secure Sockets Layer

- ▼ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
 - Content Type: Change Cipher Spec (20)
 - Version: TLS 1.2 (0x0303)
 - Length: 1
 - Change Cipher Spec Message
- ▼ TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages
 - Content Type: Handshake (22)
 - Version: TLS 1.2 (0x0303)
 - Length: 40
 - ▼ Handshake Protocol: Hello Request
 - Handshake Type: Hello Request (0)
 - Length: 0
 - ▼ Handshake Protocol: Hello Request
 - Handshake Type: Hello Request (0)
 - Length: 0

Figure: SSL/TLS Packet

TCP Handshake:

As we know XMPP uses TCP for binding we can clearly see 3-way handshake over TCP protocol between two entities to set up connection and transmit the data and end the connection.

Team14Project-capture.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp

No.	Time	Source	Destination	Protocol	Length	Info
359	631.330888	10.0.2.15	173.194.202.125	TCP		74 46423 → 5222 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
360	631.360286	173.194.202.125	10.0.2.15	TCP		60 5222 → 46423 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
361	631.360319	10.0.2.15	173.194.202.125	TCP		54 46423 → 5222 [ACK] Seq=1 Ack=1 Win=29200 Len=0

> Frame 359: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)

> Ethernet II, Src: CadmusCo_49:a5:48 (08:00:27:49:a5:48), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)

> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 173.194.202.125

▼ Transmission Control Protocol, Src Port: 46423 (46423), Dst Port: 5222 (5222), Seq: 0, Len: 0

- Source Port: 46423
- Destination Port: 5222
- [Stream index: 7]
- [TCP Segment Len: 0]
- Sequence number: 0 (relative sequence number)
- Acknowledgment number: 0
- Header Length: 40 bytes
- > Flags: 0x002 (SYN)
- Window size value: 29200
- [Calculated window size: 29200]
- > Checksum: 0x847d [validation disabled]
- Urgent pointer: 0
- > Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale

0000 52 54 00 12 35 02 08 00 27 49 a5 48 08 00 45 00 RT..5... 'I.H..E.
 0010 00 3c 83 4a 40 00 40 06 33 23 0a 00 02 0f ad c2 .<.J@.@. 3#.....
 0020 ca 7d b5 57 14 66 30 00 6c 92 00 00 00 00 a0 02 .}.W.f0. 1.....

Figure: 3-way TCP Handshake

Advantages of XMPP:

- Because of its standardized and decentralized nature of XMPP, it is very robust and powerful. It is user friendly, and if a server is turned offline only the clients that are attached to that particular server are affected.

- The architecture of XMPP is almost similar with that of e-mail. The core protocol here is to create a stream, there is no central server.
- Anybody can run his or her own XMPP server, so that individual and organizations can manage their real time messaging.
- XMPP server is independent of the public XMPP network, and safety is provided by using SASL and TLS technology
- XMPP is not only used in real time communication application but also used in network management, content syndication, collaborative tools, and file sharing, gaming and remote monitor.
- The number of available resource and support for XMPP is diverse.
- XMPP is Firewall and Nat Friendly.

Shortcomings of XMPP:

- About 60% of the data flow in the XMPP protocol is repeated. Thus XMPP has a large overhead of data to multiple recipients as compared to http
- XMPP cannot modify binary data, as it is coded as a long XML file.
- It is statefull protocol, as a result Programmers find it difficult to code.

XMPP Standards:

[RFC 3920]- This explains about the core XMPP

[RFC 6120]- This gives the specifications of the core XMPP.

[RFC 6121]- The specification regarding XMPP IM is given under this standard.

[RFC 6122]- This standard specifies the address format for the XMPP protocol.

Few Learnings (Outcomes from Project):

1. We learned the basics of XMPP protocol and its different application.
2. We have also captured the its packets and analyzed their flow using Wireshark.
3. We tested the simple architecture using Openfire and Spark client. We also analyzed Complex architecture with the help of Kraken gateway and Gtalk.
4. We also got hands on Smack APIs to communicate with Google Cloud Messaging Service XMPP endpoint and found that stanzas can also be used for passing other datatypes like JSON format.

Conclusion

The lab helped the team members in understanding various aspects of XMPP and also gave an opportunity to capture the HTTP traffic using Wireshark for further analysis. The lab helped us in learning the lab environment setup required to perform this lab using Openfire (version 4.0.2) and Spark IM Client (version 2.7.7). We as team found it challenging and interesting to explore and study XMPP, we also understood various features of Wireshark that can be used to get the flow graph, protocol hierarchy, Packet lengths etc. and Finally we were successful in producing a report about XMPP which included the summary of all the aspects that we tried and implemented during the lab execution.

By performing this experiment, we can conclude that XMPP is robust and powerful as its architecture is decentralized. It provides near real time communications through text, audio and video. Many social networking platforms uses XMPP such as Facebook, WhatsApp, Gtalk and yahoo messenger. XMPP extension protocol (XEP) provides XMPP extension like CAPTCHA, multi-user XEP, Desktop sharing and file transfer. This protocol is secured and provides authentication by using TLS and SASL protocols.

References

- [1] Download VirtualBox. (n.d.). Retrieved April 16, 2016, from <https://www.virtualbox.org/wiki/Downloads>
- [2] Download Linux Mint 17.3 Rosa. (n.d.). Retrieved April 16, 2016, from <https://www.linuxmint.com/download.php>
- [3] [RFC 6120], P.Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," Internet RFC 6120, March 2011
- [4] [RFC 6121], P.Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," Internet RFC 6121, March 2011
- [5] [RFC 6122], P.Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Address Format," Internet RFC 6122, March 2011
- [6] Peter Saint-Andre, Kevin Smith, and Remko Tronçon, "XMPP: The Definitive Guide Building Real-Time Applications with Jabber Technologies," April 2009, First Edition.
- [7] Downloads. (n.d.). Retrieved May 04, 2016, from <http://www.igniterealtime.org/downloads/index.jsp>
- [8] XMPP vs SIMPLE: The race for messaging standards. (2003). Retrieved May 04, 2016, from <http://www.infoworld.com/t/platforms/xmpp-vs-simple-race-messaging-standards-295>
- [9] (n.d.). Retrieved from <http://www.ibm.com/developerworks/library/x-xmppintro/x-xmppintro-pdf.pdf>