SERVICE SHARING MANAGER

Kuber Kaul
Masters Candidate
Columbia University
kk2872@columbia.edu

Digvijay Singh
Masters Candidate
Columbia University
ds3161@columbia.edu

ABSTRACT

The Internet was designed under the assumption that end hosts were considered stationary. With advances in wireless technology, there will be more mobile devices connecting to the Internet than fixed stationary devices. Today's mobile devices have multiple network interfaces, such as Wi-Fi, LTE, Wi-MAX, and possibly a wired LAN. Also, users are consuming Internet services from multiple devices such as smartphones, tablets, laptops, game consoles, and Wi-Fi enabled TVs. Such diverse network connectivity can be used to increase both reliability and performance, by having applications run over multiple links sequentially, and without interruption, for seamless user experience, or in parallel for bandwidth and performance enhancements. Furthermore, intelligently scheduling applications to use the appropriate network resources according to various criteria (such as cost, bandwidth, location, security, etc.) is a significant feature of this new attribute-based paradigm. Today's networking stack, however, offer almost no support to intelli- gently exploiting such network and location diversity, and no abstractions for device to device communication. The network architecture should support mobility natively, and fixed line broadband is a special case of mobile Internet.

A unified networking architecture is proposed, which makes better use of a heterogeneous dynamic environment, both in terms of networks (carrier and enterprise) and user devices. The core functionalities of the system include multihoming, mobility - terminal, network and session, multipath transport, and disruption tolerance. The system enables mobile nodes (MN) to make intelligent decisions about how and when to use each or combination of networks, in a secure manner, and based on user or network policies. With this new architecture, we envision a shift from current applications, which support a single network location, and device at a time, to applications that can support multiple networks, multiple locations, and multiple devices simultaneously or independently.

Programming Languages

The Project has been implemented in python language and the following Language Constructs and Features have been used: –

Algorithm – The algorithm we use here is basically the machine learning algorithms used to classify the movie reviews which are Naïve Bayes, KNN algorithm and the

Design – The design of the code is as follows in the order:

Cleaning of data

Classification of reviews using machine learning algorithms

Languages – The language used to implement the project is Python.

Theory – The theory of this project is loosely based on appraisal theory that is the idea that emotions are extracted from our evaluations (appraisals) of events that cause specific reactions in different people. Essentially, our appraisal of a situation causes an emotional, or affective, response that is going to be based on that appraisal. An example of this is going on a first date. If the date is perceived as positive, one might feel happiness, joy, giddiness, excitement, and/or anticipation, because they have appraised this event as one that could have positive long term effects, i.e. starting a new relationship, engagement, or even marriage. On the other hand, if the date is perceived negatively, then our emotions, as a result, might include dejection, sadness, emptiness, or fear. (Scherer et al., 2001) [1] Reasoning and understanding of one's emotional reaction becomes important for future appraisals as well. The important aspect of the appraisal theory is that it accounts for individual variances of emotional reactions to the same event

KEYWORDS

Sentiment analysis; SVM; Naïve bayes; Bag of words; KNN classification algorithm; Python; Unigrams/Bigrams/Trigrams; Feature extraction; Mutual information based selection; Frequency based feature extraction.

1. INTRODUCTION

People have become more connected than ever. Mobile communications and the Internet are at the center of this phenomenon. With more mobile devices connecting to the Internet, the bandwidth consumption of mobile broadband will soon surpass fixed line broadband [1]. In order to provide a better user experience, service providers need to handle the increase in mobile broadband traffic. However, carriers are constrained by the basic limitation of spectrum availability. Hence, there is a need for optimizing carrier resources to handle this increasing bandwidth demand.

Users expectations to communicate in this always connected world will only increase in future. Let us see how Alice would use Internet services in this more connected world of the future. Alice is on a video conference call with Bob at her office. It is getting late, she leaves her office and transfers the call to her mobile phone. She takes the subway to her home, there are no interruptions in the call when the subway is underground. Bob shares an important document with Alice and asks her to review it, as soon as Alice reaches home she transfers the video conference to her Internet enabled TV and the shared document to her desktop. In this example, we have seen how Alice wants to communicate with Bob without any interruptions and from multiple devices. To make Alices call possible, we envision a shift from current applications which support a single network, location, and device at a time to applications that can support multiple networks, multiple locations, and multiple devices requiring application device independence (session mobility), location independence (terminal mobility) and network independence (multihoming).

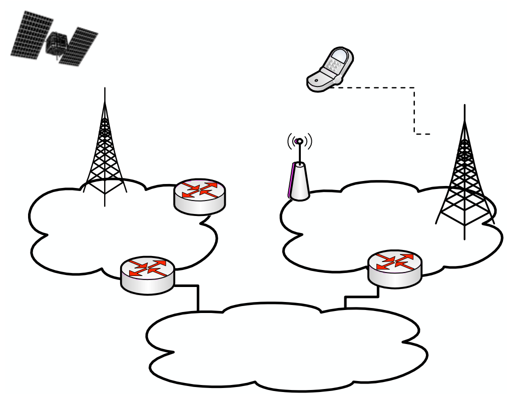
Furthermore, latest communication devices have significant mobility and processing power that provides phone, computing, video, and data communication all based upon IP protocol, for example, smartphones, tablets, and laptops. These devices have multiple network interfaces, such as Wi-Fi, WiMAX, LTE, and possibly a wired LAN. Cellular networks provide pervasive mobility with a large coverage area as compared with other networks like Wi-Fi which only provide limited coverage. Also, there are multiple service providers in the same geographic area in most of the regions around the world.

The devices may experience sequential connectivity across technologies or multiple concurrent networks. Despite this multiplicity of networks and devices, protocols are largely still assuming the use of a single network interface and address for mobile devices. For example, mobile IP is predicated on the idea of maintaining a single network address across network attachment changes, and HTTP mostly assumes a single destination address at a time. This is also true for other applications such as email clients, BitTorrent clients, etc. In fact, all applications based on the standard socket API only support one network interface at a time. For an efficient use of network resources across multiple network technologies we need an ubiquitous system where network entities (NEs) and mobile nodes (MNs) are working together in an intelligent manner to provide high throughput, resiliency, better energy management, and above all a seamless user experience for customers.

Multihoming support can offer increased bandwidth availability, reliability, flow redirection, and load balancing. better use of these multiple networks by specifying policies for network usage especially for applications which require on-demand network resources. For example, to reduce time and cost, a mobile device can use a cheaper WiFi network for downloading a file instead of using an expensive LTE connection which is always used for time critical applications like voice calls.

Mobility can be terminal, network, session, and personal [2]. Terminal mobility provides continuous network access when a MN moves from one network to another. Network mobility provides efficient network access in transportation industry, for example, WiFi enabled cars, buses, trains and more re- cently aircrafts. Having mobility support in a heterogeneous environment results in better connectivity, and cost savings for both the users and the service providers. Figure 1 illustrates a multihomed mobile environment where a MN can connect to any available network based upon its location.

Increasingly, users are consuming rich Internet services like Skype, iTunes, Google Apps from multiple mobile devices, e.g., smartphones, tablets, laptops, connected vehicles, as well as multiple IP enabled home devices. Thus, users may want to be able to move current application context from one device to another and continue work from the second device. Supporting a single application context across multiple devices requires session and personal mobility [14]. For example, Alice is watching an online video on her smartphone, she can seamlessly transfer the application session along with her credentials to another device and resume the video where she left it.



Several solutions have been proposed which decouple location and identification of end nodes to support multi-homing and mobility. To the best of our understanding, there has not been any proposal considering the combination of multi-homing, terminal mobility, session mobility and disruption tolerance as the basic abstractions for a new network architecture. In this paper, we first summarize and evaluate presently available relevant solutions in view of our goals, and propose what is needed to achieve the vision of a unified Internet architecture. In Section II, we identify the design goals that are essential for our vision. In Section III, we survey some of the important proposals related to future Internet architecture functionalities in view of their support for multihoming, mo- bility (terminal, session) and disruption tolerance. In Section IV, we propose our initial design of the architecture. Section V discusses the future vision of the proposed architecture.

1.1 *Normal Background and Related Work*

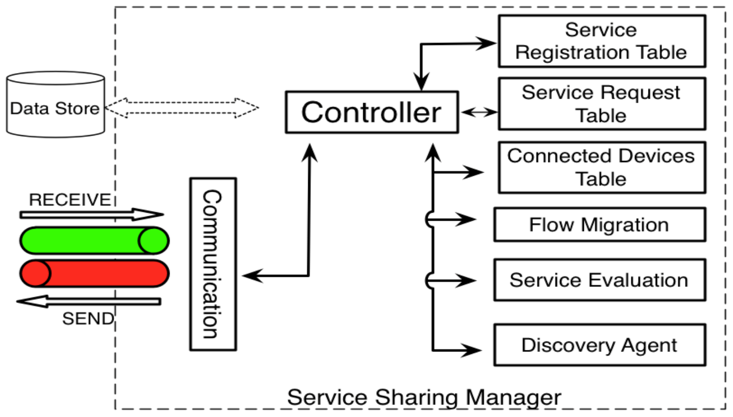
Service sharing has been the focus of recent research. It has been attempted in different domains such as audio over RTP, video over RTP, webcam over RTP, transfer of media packets relaying over devices. We can thus say that Sentiment classification has indeed become very important topic of research. (Pang et al., 2002; Turney and Littman, 2003; Pang and Lee, 2004; Beineke et al., 2004; Gamon, 2004). We already have developed a vertical network protocol stack into which we would be integrating our service sharing manager. The control middleware provides a general platform for service sharing across multiple devices in a secure trusted environment. The service-sharing manager (SSM) allows the control middleware to create new flows between devices and modify existing flows by adding / removing devices.

1.2 *Service Sharing Manager*

The SSM enables service sharing in a trusted networking environment using new service sharing APIs. The SSM module provides three kinds of service: (1) service registration, (2) service request and (3) flow migration. Figure 1 illustrates the internal components of a SSM. The discovery agent (DA) is responsible for service discovery in the network. The DA's discovery protocol is configurable; it can use Service Location Protocol (SLP), Simple Service Discovery Protocol (SSDP) or any other service discovery protocol based upon the implementation. The service evaluation (SE) component evaluates (scores) the service based upon the service request. The Flow Migration (FM) component performs session mobility from one device to another. The two tables: Service Registration and Service Request are used for maintaining and monitoring active services in the network. The connected devices table maintains the active device list with the corresponding location details (IP address, network SSID, device type, etc.) of the device.

The SSM enables service sharing in a trusted networking environment using new service sharing APIs. The SSM module provides three kinds of service: (1) service registration, (2) service request and (3) flow migration. Figure 1 illustrates the internal components of a SSM. The discovery agent (DA) is responsible for service discovery in the network. The DA's discovery protocol is configurable; it can use Service Location Protocol (SLP), Simple Service Discovery Protocol (SSDP) or any other service discovery protocol based upon the implementation. The service evaluation (SE) component evaluates (scores) the service based upon the service request. The Flow Migration (FM) component performs session mobility from one device to another. The two tables: Service Registration and Service Request are used for maintaining and monitoring active services in the network. The connected devices table maintains the active device list with the corresponding location details

(IP address, network SSID, device type, etc.) of the device.



1.3 *Service Discovery*

To share data and services in a disconnected network mobile users have to rely on node and service discovery methods. Moghadam et al. [27] present a modular application development framework denominated as seven degrees of seperation (7DS) to support opportunistic communication in a disconnected dynamic environment. 7DS provides the necessary transport and application layer functionalities for mobile nodes to exchange information in store-carry-forward manner. In our project we also had to integrate service discovery into our application. Service discovery has been researched on for quite some time and there are certain service discovery techniques that are quite prevalent and we went through most of them to review it for our project. Service discovery is the ability of our devices to identify services/other devices in the network and then the respective software will help establish connections between them to finally push packets (audio/video) between those devices to create the regular streaming without inputting any IP as we know it.

The various techniques of service discovery that we went through and implemented for review in our devices were dns-sd, zero configuration implementations of avahi and bonjour both. To test these implementations we tested them on an iphone app called discovery and then firefox and chrome dns-sd plugin to detect if these were working as per the expectations.

1.4 *DNS-SD*

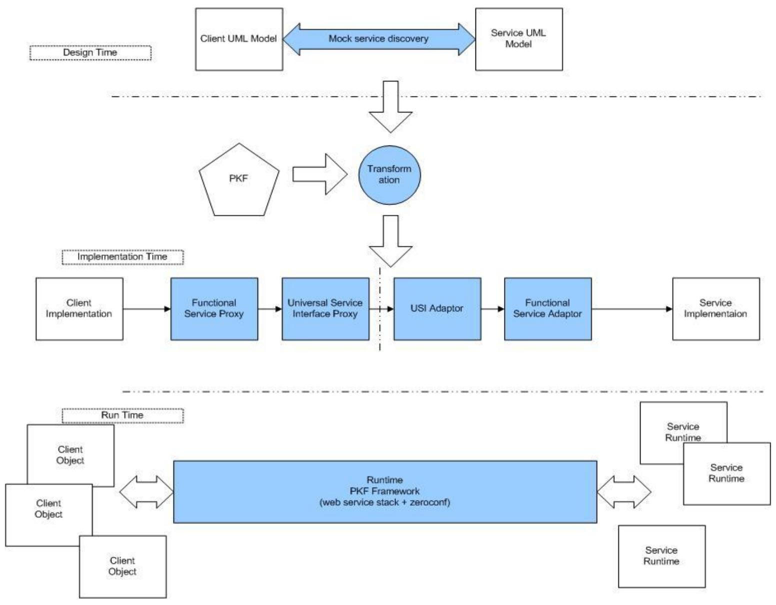
- DNS Service Discovery is a way of using standard DNS programming interfaces, servers, and packet formats to browse the network for services.It is one of the techniques of implementing zero conf networking.
- Given a type of service that a client is looking for, and a domain in which the client is looking for that service, this allows clients to discover a list of named instances of that desired service, using standard DNS queries.

Three requirements for zero configuration networks:

1. IP address assignment without a DHCP server
2. Host name resolution without a DNS server
3. Local service discovery without any rendezvous server

Solutions and implementations:

- RFC3927: Link-local addressing standard for 1)
- DNS-SD/mDNS: Apple’s protocol for 2) & 3)
- Bonjour: DNS-SD/mDNS implementation by Apple
- Avahi: DNS-SD/mDNS implementation for Linux and BSD



1.5 *Zero Configuration*

Some major points about zero-configuration:

- Zeroconf Working Group of IETF (The Internet Engineering Taskforce) since 1999
- Goals of zero configuration:
- Allocate addresses without a DHCP server.
- Translate between names and IP addresses without a DNS server.
- Find services, like printers, without a directory server.

Zero configuration has two major implementations for service discovery which we tested for our application, those being Bonjour and Avahi. Service Discovery will use zeroconf to be automatic. Zeroconf also serves as a distributed directory service. Within the same type of services, the “Service Info” of each service will contain the service category and subcategory information. The term distribution is used to indicate that multiple directory servers, that hold different namespaces, are interconnected to form a distributed directory service.Service addition and removal is automatically reported to all clients who are interested in the same type of service.

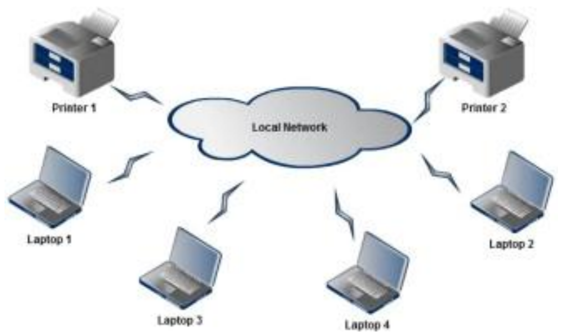
1.5.1 *Bonjour – Zero configuration implementation*

Bonjour is the zero configuration implementation created by apple for there products.It is not open source hence limited for usage but was the first DNS-SD implementation with widespread use in the apple products and was the adaptation from Rendezvous from Apple Inc., uses multicast DNS and DNS Service Discovery. Apple changed its preferred zeroconf technology from SLP to mDNS and DNS-SD between Mac OS X 10.1 and 10.2, though SLP continues to be supported by Mac OS X.

Apple's mDNSResponder has interfaces for C and Java[21] and is available on BSD, Apple Mac OS X, Linux, other POSIX based operating systems and MS Windows. The Windows downloads are available from Apple's website.We were successful in able to detect services and devices in the network through bonjour.

1.5.2 *Avahi – Zero configuration implementation(Linux)*

Avahi is the zero configuration implementation in Linux and is an open source software. It has and can perform everything that bonjour can but the only problem is the documentation which is now increasing day by day and will soon match the documentations of bonjour for easy and systematic usage.



Avahi implements the Apple Zeroconfiguration specification, mDNS, DNS-SD and RFC 3927/IPv4LL. Other implementations include Apple's Bonjourframework (the mDNSResponder component of which is licensed under the Apache License).

Avahi provides a set of language bindings (Python, Mono, etc.) and ships with most Linux and *BSD distributions. Because of its modularized architecture, major desktop components like GNOME's Virtual File System and the KDE input/output architecture already integrate Avahi.

Avahi performed as per our expectations and being an open source software it was easy to get all parts working with our implementation. Though, there was a lack of documentation but with thorough research and whatever documentation we could get we managed to get it working as per our liking.

1.5.3 *UPnP – Universal Plug'n'Play*

Universal Plug and Play is the peer-to-peer network connectivity of intelligent appliances, wireless devices and PCs of all forms. It is Zero-configuration, flexible networking and has standard-based connectivity to ad-hoc or unmanaged networks. Upnp works on distributed systems as well and hence that helps in detecting devices/services on different systems.

It Reduces footprint i.e there is no trace left after the handover takes place. UPnP is OS, language and media independent and make use of IP, TCP, UDP, HTTP and XML in its implementation. It is still not as advanced as UPnP even though it was widely used once.

A UPnP AV media server is the UPnP-server (a 'master' device) that provides media library information and streams media-data (like audio/video/picture/files) to UPnP-clients on the network. It is a computer system or a similar digital appliance that stores digital media, such as photographs, movies, or music and shares these with other devices.

UPnP AV media servers provide a service to UPnP AV client devices, so called control points, for browsing the media content of the server and request the media server to deliver a file to the control point for playback.

UPnP media servers are available for most operating systems and many hardware platforms. UPnP AV media servers can either be categorized as software-based or hardware-based. Software-based UPnP AV media servers can be run on a PC. Hardware-based UPnP AV media servers may run on any NAS devices or any specific hardware for delivering media, such as a DVR. As of May 2008, there were more software-based UPnP AV media servers than there were hardware-based servers.

2. *Packet Transmission using RTP*

To transmit an RTP stream, we used a Processor to produce an RTP-encoded DataSource and construct either a SessionManager or DataSink to control the transmission.

The input to the Processor can be either stored or live captured data. For stored data, you can use a MediaLocator to identify the file when you create the Processor. For captured data, a captureDataSource is used as the input to the Processor, as described in Capturing Media Data.

There are two ways to transmit RTP streams:

- Use a MediaLocator that has the parameters of the RTP session to construct an RTP DataSink by calling Manager.createDataSink.
- Use a session manager to create send streams for the content and control the transmission.

In case of MediaLocator to construct an RTP DataSink, you can only transmit the first stream in the DataSource. If you want to transmit multiple RTP streams in a session or need to monitor session statistics, you need to use the SessionManager directly.

Regardless of how we choose to transmit the RTP stream, we need to:

1. Create a Processor with a DataSource that represents the data you want to transmit.
2. Configure the Processor to output RTP-encoded data.
3. Get the output from the Processor as a DataSource.

2.1 Configuring the Processor

To configure the Processor to generate RTP-encoded data, we set RTP-specific formats for each track and specify the output content descriptor you want.

The track formats are set by getting the TrackControl for each track and calling setFormat to specify an RTP-specific format. An RTP-specific format is selected by setting the encoding string of the format to an RTP-specific string such as "AudioFormat.GSM_RTP". The Processor attempts to load a plug-in that supports this format. If no appropriate plug-in is installed, that particular RTP format cannot be supported and an UnSupportedFormatException is thrown.

The output format is set with the setOutputContentDescriptor method. If no special multiplexing is required, the output content descriptor can be set to "ContentDescriptor.RAW". Audio and video streams should not be interleaved. If the Processor's tracks are of different media types, each media stream is transmitted in a separate RTP session.

2.2 Retrieving the Processor Output

Once the format of a Processor's track has been set and the Processor has been realized, the output DataSource of the Processor can be retrieved. You retrieve the output of the Processor as a DataSource by calling getDataOutput. The returned DataSource can be either PushBufferDataSource or a PullBufferDataSource, depending on the source of the data.

The output DataSource is connected to the SessionManager using the createSendStream method. The session manager must be initialized before you can create the send stream.

If the DataSource contains multiple SourceStreams, each SourceStream is sent out as a separate RTP stream, either in the same session or a different session. If the DataSource contains both audio and video streams, separate RTP sessions must be created for audio and video. You can also clone the DataSource and send the clones out as different RTP streams in either the same session or different sessions.

2.3 Controlling the Packet Delay

The packet delay, also known as the packetization interval, is the time represented by each RTP packet as it is transmitted over the network. The packetization interval determines the minimum end-to-end delay; longer packets introduce less header overhead but higher delay and make packet loss more noticeable. For non-interactive applications such as lectures, or for links with severe bandwidth constraints, a higher packetization delay might be appropriate.

A receiver should accept packets representing between 0 and 200 ms of audio data. (For framed audio encodings, a receiver should accept packets with 200 ms divided by the frame duration, rounded up.) This restriction allows reasonable buffer sizing for the receiver. Each packetizer codec has a default packetization interval appropriate for its encoding.If the codec allows modification of this interval, it exports a corresponding PacketSizeControl. The packetization interval can be changed or set by through the setPacketSize method.

For video streams, a single video frame is transmitted in multiple RTP packets. The size of each packet is limited by the Maximum Transmission Unit (MTU) of the underlying network. This parameter is also set using the setPacketSize method of the packetizer codec's PacketSizeControl.

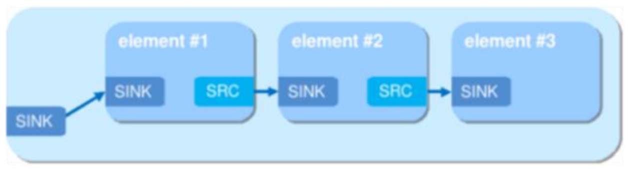
2.4 *Transmitting RTP Data With a Data Sink*

The simplest way to transmit RTP data is to construct an RTP datasink using the manager. Create Data Sink method. You pass in the output DataSource from the Processor and a MediaLocatorthat describes the RTP session to which the DataSource is to be streamed.
(The MediaLocator provides the address and port of the RTP session.)

To control the transmission, you call start and stop on the DataSink. Only the first stream in the Data Source is transmitted.

3. Routing of Packets – Our design

Our design of the routing of packets be it audio or video was to create a UDP server on source device and then create a UDP sink on every recipient device in the chain of devices. We need to have gstreamer and the required components installed in every device. We also install avahi in every system. All this is integrated together in a neat user interface by us. Now the server starts encoding the audio or video files from their respective formats to a RTP stream which is then pushed from server to the next device(sink) in the form of RTP stream using gstreamer. This gets saved in the next device, which will then simultaneously read from this newly created file and push it further to multiple devices till it reaches the final sink. The two processes in the devices between source and sink are run almost simultaneously resulting in the real time movement of packets. There might be the case of loss of packets or lag which is taken care by our code which implements a buffer which in turn streams packets only after a certain threshold is reached and results in perfect delivery of packets.



4. Gstreamer

We have used studied and used gstreamer in-depth for our project demonstration :

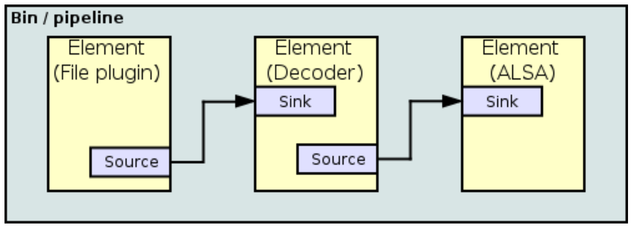
GStreamer is a pipeline-based multimedia framework written in the C programming language . GStreamer allows a programmer to create a variety of media-handling components, including simple audio playback, audio and video playback, recording, streaming and editing. The pipeline design serves as a base to create many types of multimedia applications such as video editors, streaming media broadcasters and media players. Designed to be cross-platform, it is known to work on Linux, Solaris (Intel and SPARC) and OpenSolaris, FreeBSD, OpenBSD, NetBSD, MacOSX, Microsoft Windows and OS/400.

GStreamer has bindings for programming-languages like Python, Vala, C++, Perl, GNU Guile and Ruby. GStreamer is licensed under the GNU Lesser General Public License. For our demonstration we used python language bindings of Gstreamer. Since it a scripting language and in use for our current courses it was the first language to go to. We have created separate command line arguments for the various streaming tasks and the routing tasks written with gstreamer pipelines implemented. We also coded the same thing in python for a functional and independent module to be used in and with SINE.

GStreamer allows a programmer to create a variety of media-handling components, including simple audio playback, audio and video playback,recording, streaming and editing. The pipeline design serves as a base to create many types of multimedia applications such as video editors, streaming media broadcasters and media players. We used this property to the hilt by creating gstreamer pipelines to modulate the audio and video streams so as to save them or just stream them as per our liking.

Gstreamer can be used for various other purposes of handling media components like creating a media player, audio editors, video editors, using RTSP of bring controls in the streaming. GStreamer processes media by connecting a number of processing elements into a pipeline. Each element is provided by a plug-in. Elements can be grouped into bins, which can be further aggregated, thus forming a hierarchical graph. This is an example of a filter graph. Elements communicate by means of pads. A source pad on one element can be connected to a sink pad on another. When the pipeline is in the playing state, data buffers flow from the source pad to the sink pad. Pads negotiate the kind of data that will be sent using capabilities.

The diagram below could exemplify playing an MP3 file using G-Streamer. The file source reads an MP3 file from a computer's hard-drive and sends it to the MP3 decoder. The decoder decodes the file data and converts it into PCM samples which then pass to the ALSA sound-driver. The ALSA sound-driver sends the PCM sound samples to the computer's speakers.



GStreamer uses a plug-in architecture which makes the most of GStreamer's functionality implemented as shared libraries.[11]GStreamer's base functionality contains functions for registering and loading plug-ins and for providing the fundamentals of all classes in the form of base classes. Plug-in libraries get dynamically loaded to support a wide spectrum of codecs, container formats, input/output drivers and effects.

Plug-ins can be installed semi-automatically when they are first needed. For that purpose distributions can register a backend that resolves feature-descriptions to package-names.

Since version 0.10, the plug-ins come grouped into three sets (named after the film The Good, the Bad and the Ugly),

Plug-in set name	Description
Good	This package contains the GStreamer plug-ins from the "good" set, a set of high quality plug-ins under the LGPL license,[12]or according to Gstreamer, "contains a set of well-supported plug-ins under our preferred license".[13]
Bad	GStreamer Bad Plug-ins comprises a set of plug-ins not up-to-par compared to the rest. They might closely approach good-quality plug-ins, but they lack something: perhaps a good code review, some documentation, a set of tests, a real live maintainer, or some actual wide use.[14]
Ugly	This package contains plug-ins from the "ugly" set, a set of good-quality plug-ins that might pose distribution problems.[15]

We have created various audio/video modulation pipelines of gstreamer for our use to modify/save/stream the RTP packets/stream as per the requirements.We have created a separate paper on gstreamer pipelines and python code for our individual programs as we think due to the lack of documentation on the subject those might be a useful open source resource for other people working with gstreamer. All pipelines can be found there along with the code that we have submitted.

5. *Acknowledgement*

We would like to sincerely thank Dr. Gaston Ormazabal who is a Senior Research Scientist at Columbia University in the Department of Computer Science co leading the SINE project. We would also like to thank Mr. Aman Singh who is a graduate student of Columbia University and is currently working with Professor Henning Schulzrinne at the internet real time lab. Lastly, we also want to acknowledge Professor Henning Schulzrinne, professor in the Dept. of Computer Science; also with the Dept. of Electrical Engineering for his research work and creation of RTP protocol which we have immensely used in our project. All of the above people acted as guides for us to complete this project to satisfaction.

6. *References*

a. Gstreamer : http://gstreamer.freedesktop.org/
b. Upnp : http://www.upnp.org/
c. DLNA : http://www.dlna.org/
d. Gstreamer SDK : http://www.gstreamer.com/
e. RTP : http://www.cs.columbia.edu/~hgs/rtp/
f. Stack Overflow : http://stackoverflow.com/
g. RTP support in Gstreamer: http://gstreamer.freedesktop.org/documentation/rtp.html
h. Gstreamer plugins : http://gstreamer.freedesktop.org/documentation/plugins.html
i. UDP : http://tools.ietf.org/html/rfc768

j. ZeroConfig : http://zeroconf.sourceforge.net/

k. Gstreamer pipelines : Filepiper (open source – Aman Singh)

l. *GStreamer*: Documentation - gstreamer.freedesktop.org/documentation/

m. GStreamer: features - gstreamer.freedesktop.org/features/

n. UPnP Hacks: Hacking Universal Plug and Play - www.upnp-hacks.org/

o. libupnp.org - Developer resources for the portable UPnP™ library - pupnp.sourceforge.net/

p. GStreamer - The GNOME Development Site - developer.gnome.org/platform-overview/stable/gstreamer

q. avahi-daemon(8): Avahi mDNS/DNS-SD daemon - Linux man page - linux.die.net/man/8/avahi-daemon

r. “avahi” package : Ubuntu - https://launchpad.net/ubuntu/+source/avahi

s. RTP, Real-time Transport Protocol - www.networksorcery.com/enp/protocol/rtp.htm

t. RFC 768 - User Datagram Protocol - tools.ietf.org/html/rfc768

u. Gstreamer - Wikipedia.com/Gstreamer