PDS: Video Streaming/File Sharing

Badrinath Radhakrishnan(110815408)*, Karan Malhotra(111016411)*
Department of Electrical and Computer Engineering, Stony Brook University,
Stony Brook, NY
Email: *badrinath.radhakrishnan@stonybrook.edu, *karan.malhotra@stonybrook.edu

***Abstract*—We propose a very novel system, an application that will enable people to live stream the content that is made available among different peer groups that are connected in the same network. Similarly, the same functionality can further be extended that will allow data sharing among the various members who are connected in the same network. This is also called as peer to peer sharing. With this application, people will not have to worry about any internet connection or Wifi connection to share data/information that might prove useful for general public. For example, to provide a concise idea of what will be the traffic conditions ahead, the provider present at the live scene might provide a live video feed that can be streamed on the go by different users. The different users present in the same network will then be able to figure out an alternate way, in case the traffic situation currently is beyond being able to improve in the near time. The base over which this application has been built and which provides it the networking abilities to share data among different groups of peers, acting as either a sender or a recipient of data is the PEC Service. The PEC Service consists of API packages that are utilized to build applications which exploit all the potential of the service.**

I. INTRODUCTION

A. MOTIVATION

Imagine, one day when suddenly moving down the road, you see an ambulance trying to make its way ahead from all other vehicles with the best of its capability, but still is unable to do so due to huge traffic chaos. A very innovative and advanced use of such an application that we are proposing here, would be to enable video streaming in real time and make it available for all the other users as well whom we want the message to be conveyed. This kind of data sharing is known as peer to peer sharing. A lot of applications(e.g., SHAREit) that enable us to transmit data from one place to another, do exist in today's world but all of them have their own flaws. But with the application that we are trying to build here, tries to address all those flaws and carefully provide a seamless way of transporting data from one device to another. Thus, it provides a clear passage of data transfer and in doing so, we don't require any kind of internet connection as well. However, there are various challenges still involved prior to the suitable implementation of this system. One of the challenge that we face is to enable users and providers to get connected to each other without being present in the same network. This concept then will apply to each any everyone who wants to be connected, to one another, for data sharing at their own will. This will extricate probably all the constraints that many other data sharing applications provide. So, the whole idea of simply sharing or providing the live feeds or data makes it quite a unique one. Observing the proliferation and daily congregations of modern mobile devices that has created abundant opportunities for peer users to share valuable data with each other, the idea came into existence. A design called PDS has been proposed which is a robust, efficient peer data discovery and retrieval among Mobile Devices. Peer Data Sharing (PDS) enables devices to discover which data exist in nearby peers, and retrieve interested data robustly and efficiently. The service over which this application has been built is known as PEC Service. This service enables sharing data and resources among opportunistically gathered mobile devices. Some of the features include: peer based service, doesn't require any centralized control, i.e. the participants decide how much they are willing to participate.

B. CHALLENGES AND POSSIBLE SOLUTION

With the features and ideas mentioned above, as well as the the needs this application has the potential to cater to, we believe working on building such a system is quite worthwhile. The existing mechanism for peer communication such as WiFi ad-hoc mode has the undesirable consequence of disconnecting users from the access points and Internet.And thus, building such an application that removes the above mentioned restrictions make the whole system quite unique from the other existing systems. However, since the nature of such a project we have been working on is research type, this system requires continuous improvements as and when the software updates are made. Also, the PEC service is based on an idea called UDP broadcasting which is still not a totally reliable method as the packets of data may get lost while getting transferred from one device to another and then the same packet needs to be transferred again. The best possible way to tackle such a problem is to divide the entire data into number of smaller chunks as the smaller the chunk size, the more the probability that the chunk will be transferred successfully. Another advantage of dividing the entire data into number of chunks is that, if by chance a particular chunk does not able to get transferred, then we don't need to transfer the entire data again but from that particular chunk that got missed initially. Thus, this technique saves us a lot of time while transferring the data. However, it should be noted that there is still a scope of improving the speed of data transfer to a much greater value than the current existing speed, by making further improvements in design of the service.

II. PROPOSED APPROACH

We try to build an application that is built over a service that provides the networking capabilities of sharing useful data among the opportunistically gathered mobile devices. The service over which our system works has been designed with an idea of returning all data existence information faithfully, despite dynamic changes in both device and data sets. It uses lingering queries to retrieve continuous streams of returning meta-data entries, mixedcast for efficient data delivery to multiple consumers, and en-route message rewriting to minimize redundant meta-data collection. It is designed as a two-phase data retrieval service that retrieves different portions of data from multiple nodes robustly and efficiently and gathers chunk distribution information on demand. The basic and one of the most important concept involved here is that of data descriptors. Each descriptor is a metadata entry that indicates the possible availability of the corresponding data item/chunk. Thus all such entries together describe what data may exist in the network. Because metadata entries have small sizes and are frequently requested by many consumers, they are widely cached. Any node receiving, relaying or overhearing metadata entries will cache them to serve potential future requests. After different providers from different locations register into the data descriptors, they will be able to send the data and we use this scheme to get the data into the meta-data list. The entire design consists of integrating this application with the PEC service so that a live video flow path/file sharing can be obtained, obtaining the video streams from the provider and providing it to the users for successful viewing of the live feeds. For video streaming, since multiple providers are capable of providing the live stream, so in order to segregate each provider from another, we will map their location with the help of co-ordinates stored in the data descriptors.

It is worth mentioning that the service has been built considering some of the assumptions that are mentioned as follows: First, the environment is uncertain and dynamic. And since this application has been built with an idea of being centralized, that is not granting control to only some particular users, the congestion of the network may vary from place to place(for e.g., more number of users might be connected in places like parks, schools and colleges, than other places). Second, the devices have reasonable storage (e.g., 16GB or higher) and they can cache others data, both relayed or overheard. Each device can be a consumer that requests desired data, or a producer that provides them (either generated locally or cached). And typically two scenarios may exist: the consumer needs many small data items meeting certain criteria (e.g., air pollution samples in certain area), or one large, possibly popular data item (e.g., a video clip) consisting of many small chunks, each available from multiple nearby devices. Some of the points are worth mentioning here:

- On any device, a metadata entry exists as long as the corresponding data item (or any chunk of the data item) exists. That is why, this system that is built over the PEC service, is an on-the-go system. The user will be able to stream the live feed as long as the feed is available in the meta-data list.
- Peer Data Sharing (PDS) consists of two components: Peer Data Discovery (PDD) and Peer Data Retrieval (PDR). They share similar message formats, processing procedures and routing mechanism. PDD collects metadata through multi-round requests. In each round the consumer sends a query message requesting metadata, and waits for response messages carrying metadata entries to return. The consumer dynamically decides whether and when to start a new round, or terminate the data discovery if almost all data entries are returned. In each round, a consumer propagates a metadata query throughout the network. Each node receiving that query should reply all the metadata entries it holds. Those metadata will be delivered back to the consumer along the reverse path of query propagation.
- The WiFi radio works in infrastructure mode. Thus the users normal Internet connection and activities are not affected. To enable overhearing, nodes send all PDS messages in UDP broadcast. This way, all neighbors can overhear the message and choose to cache any contained data when needed.

III. DESIGN

The video/File streaming application that has been developed is built on top a service called PEC. PEC aids in sharing of data and resources among opportunistically gathered mobile devices.On e important feature of PDS is that it is peer based i,e, it does not require any cloud end server to be maintained and there is no centralized control.

A. PECSERVICE API

PDS contains a list of API's that it used to transmit and receive the data and resources between various devices. Those are **APP-MSG-REGISTER-APP**,**SRV-MSG-REQUEST METADATA**, **APP-MESSAGE-PROVIDE METADATA**, **SRV-MSG-REQUEST-METADATA**,**APP-MSG-PROVIDE-DATA**, **APP-MSG-REQUEST-DATA**, **SRV-MSG-PROVIDE-DATA**.Initially when the mobile device is connected to the service the device uses the API APP-MSG-REGISTER-APP to indicate its desire to get registered to the service.After registering to the service the device has two options: Either to act as a provider to provide data or as an user to receive the data that is currently available.The provider who wishes to share the data adds the file details or the video details to the descriptor which is an key value object that contains details about the file being shared such as file name and in case of video streaming it contains details regarding the video such as the GPS coordinates of the location from which the video is being streamed from. If the video is being streamed from the library it contains the the coordinates of the library location. The provider also adds the details of the video or the file stream to the meta data descriptor through the API APP-MESSAGE-PROVIDE METADATA. The metadata is

similar to a database which contains the list of files that are currently available in the buffer that can be downloaded by all other users that are connected to the service or it may contain the details of the video that was added to the descriptor. It is this metadata that helps users to identify the videos or the files which they wish to view or download respectively. In order to see the list of data that are currently available and can be downloaded or viewed that user must first request the service to provide the device with a list of metadata that are currently available. For the the user requests the service using the API APP-MSG-REQUEST-METADATA.On listening to this request the service requests the provider to provide it with a list of metadata. This is done with the help of the API SRV-MSG-REQUEST-METADATA. Once the service requests for the metadata the provider responds by providing the service with the required metadata using the API APP-MSG-PROVIDE-METADATA.Upon obtaining the required metadata from the provider the service then provides the metadata that the user had requested for through the API SRV-MSG-PROVIDE-METADATA.Using this list of metadata the user device extracts the details of the video stream that are currently available for viewing such as stream location and file names for the files that are currently available for download.The user then selects from the list the video he wishes to view or the file he wishes to download. For this the user requests the service through the API APP-MSG-REQUEST-DATA along the details of the video or the file which he wishes to view or download respectively. The service then matches the details that the user provided with the appropriate descriptor in order to locate the correct provider.The service then requests the provider to provide the data to the service through the API SRV-MSG-REQUEST-DATA.For this the provider responds with the data to be provider through the API APP-MSG-PROVIDE-DATA.Here the data corresponds to the stream of bytes that are extracted from the video or the file content. The service then accepts the data provided by the provider and then provides it to the user who had earlier requested for the data through the API SRV-MSG-PROVIDE-DATA.This data is then rendered on the users device through suitable mechanism. The service then continues to provide data as long as the video is available for viewing or file is available for download.

B. DATA COMPRESSION

The data that is to be transmitted is first compressed.The main purpose of compression algorithm is to reduce the data size that is being transmitted from the provider to the user device.In this application it is compressed using a java class called Deflater. The Deflater accepts the byte stream produced by the video stream or the file content as input and feeds it to the zlib compression library.zlib is a software library used for data compression.It uses Huffman coding and LZ77 compression algorithms. The byte stream that is fed is divided into blocks and each block uses single mode of compression.Deflater consists of several type of modifiers based on which the compression is performed varying from best compression to no compression. The video streaming and file sharing application that was developed uses the filtered compression which is particularly good for data consisting mostly of small values.Once the data is compressed it is returned from the function as an ByteArrayOutputStream. This data is then transmitted form the device via the PEC service.

On the receiver end the bytestream that the service provides is fed to a decompression algorithm. Here we use a class called Inflater.Inflater class provides support for general purpose decompression using the popular ZLIB compression library. The Inflater accepts the byte stream that the service provides and then feeds it to the zlib compression library.The zlib library then converts the bytestream back to the original state similar to the deflate algorithm by converting the data into blocks and then applying the decompression to individual blocks of data.These blocks are then combined to form the original ByteArrayOutputStream.

C. SURFACEVIEW

The video that is being streamed is drawn on an surface embedded inside of a view hierarchy. This surface is called the Surface view.The SurfaceView punches a hole in its window to allow its surface to be displayed and it is Z ordered, i.e, buttons that control the playing of the videos can be placed on top of the surface view.Access to the underlying surface is provided via the SurfaceHolder interface.In order to use the surface we must implement the surfacecreated and surfacedestroyed API to discover when the Surface is created and destroyed as the window is shown and hidden. One of the major advantage of using surface view is that it provides a surface in which a secondary thread can render into the screen.This reduces the significant overhead that video streaming might cause on the main thread.Another API that must be used is the surfacechanged function.This is called immediately after any structural changes (format or size) have been made to the surface

D. CAMERA API

In order to record the videos we have used the camera API that is available in android.Before using this we have to request the user to provide permission to use the camera. We do this by declaring the use of camera feature in androidManifest.xml file.In order to start recording video using the camera it is necessary to call the following functions. Camera.open(),Camera.setPreviewDisplay(), and Camera.startPreview().The camera .open method is used to get an instance of the camera object. Camera.setPreviewDisplay method is used to Prepare a live camera image preview by connecting the camera to the surface view SurfaceView.Camera.startPreview API is used to begin displaying the live camera images. When the camera is being used with the media recorder it is necessary to lock the camera unlock the camera for the mediarecorder to use the camera and then lock it so that other applications cannot request for the camera use while the mediarecorder is recording videos.Once the camera activity is finished it is necessary to stop the preview by calling

stoppreview API.Then the camera instance must be released for other application to use it

E. MEDIA RECORDER

Another important API that this application uses is the mediaRecorder.Media recorder is used to record audio and video.The recording control is based on a simple state machine that is shown in Figure 1

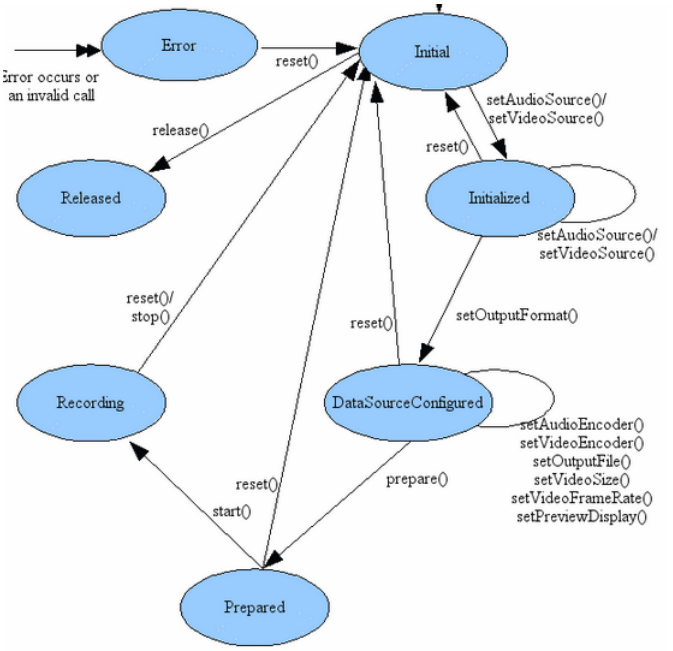

Fig. 1. Media Recorder State diagram

In order to use the media recorder we have to use the media recorder instance.Its syntax is MediaRecorder myAudioRecorder = new MediaRecorder();.It also accepts several other options such as videos source, output video format,output file location,setcamera,video frame rate,and the encoder.Once all the parameters are set we then call the prepare api which prepares the recorder to begin capturing and encoding data.Finally we call the start api to start recording. As media recorder does not allow users to directly extract the byte stream while recording,in order to extract the byte stream from the media recorder we have to do some hack. We have to call the setpreviewcallback method of the camera instance once the surface has changed and also after calling the start API of the media recorder.Then we call the setPreviewDisplay and startPreview methods of the camera instance. This enables us to extract the byte stream while the video is getting recorded. This byte stream will be available in the setpreviewcallback method.We then use this byte stream send it to the service which had requested for it. The byte stream that was generated will be in YUI format. we can then convert it into RGB format for drawing on the surface view. Once the recording is over it is necessary to release the media recorder and the camera instance to prevent any sort of errors that might occur. This is because ,the camera instance after it has been created and set to the mediarecorder , it is locked. i,e. it cannot be used by other application that requests for it. Hence it is necessary to release it once the recording is over for other applications to use it if required.

F. LOCATION SERVICE

The video streaming application uses the location service to determine the coordinate of the location from where the video is being streamed from.For the application to use the GPS, it is mandatory to get the permission from the user for the app to allow location access. In order to receive location updates from NETWORK-PROVIDER or GPS-PROVIDER, you must request the user's permission by declaring either the ACCESS-COARSE-LOCATION or ACCESS-FINE-LOCATION permission, respectively, in your Android manifest file. Without these permissions, your application will fail at run-time when requesting location updates.

If you are using both NETWORK-PROVIDER and GPS-PROVIDER, then you need to request only the ACCESS-FINE-LOCATION permission, because it includes permission for both providers. Permission for ACCESS-COARSE-LOCATION allows access only to NETWORK-PROVIDER. The location listener is used for receiving notifications from the LocationManager when the location has changed.Any class implementing the location listener has to override the following methods *onLocationChanged*: It is called when the location has changed. *onProviderDisabled*: It is called when the provider is disabled by the user. *onProviderEnabled*: It is called when the provider is enabled by the user. *onStatusChanged*: It is called when the provider status changes.

G. ASYNC TASK

In order to calculate the GPS co-ordiates of the location from which the video is being streamed we extend the Async task class.Async Task enables proper and easy use of the UI thread.This class allows us to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers. Here the background operation that we are doing is calculating the GPS coordinates.AsyncTasks should ideally be used for short operations .An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

onPreExecute(), is invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance this can be used to display progress bar that indicates percentage of the file that has been downloaded.

doInBackground(Params...), is invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are

passed to this step.We use this to map the coordinates that we obtained from the Geolocater to the street address, city and state of the place.This can be used to map the coordinates that we obtain from the meta data to display the location in a more readable form.

onProgressUpdate(Progress...), is invoked on the UI thread after a call to publishProgress(Progress...). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing.For example , this method could help identify the amount of data that has been received in case of file sharing and hence update the progress bar.

onPostExecute(Result), is invoked on the UI thread after the background computation finishes.The result of the Geolocation mapping calculated in the doInBackground method is passed on to this step and these values can be set in the viewing area of the user device. The class which extends the Async task is invoked calling the execute function on the main thread.The parameters to the AsyncTask class is passed via the execute method.

H. SERVICE

A Service is an application component that represents either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use.In order for an application to use the service it should have a corresponding ¡service¿ declaration in its package's AndroidManifest.xml.The service that the application wants to use can be started with with Context.startService() and Context.bindService().The startService method provides a request to start the application service. It takes an Intent as it parameter. For this the intent should contain either the complete class name of a specific service implementation to start or a specific package name to target.The bindservice API defines a dependency between the application and the service. It takes three parameters. First is the intent which is the name of the service to be connected to, second is the serviceconnection which receives information as the service is started and stopped.Third is the flag which specifies operation options for the binding. The the video streaming and file sharing application that we have built connects to a service class called the PECService. the flag that we use is BIND-IMPORTANT. By this we mean to say that the PECService is very important and so should be brought to the foreground process level when the client is.The system attempts to keep running services around as much as possible.Once the service is bound the client receives the IBinder object that the service returns which allows the client to make calls back to the service.The state of the application service is monitored by the service connection.It has two methods namely onServiceConnected and onServiceDisconnected. The onServiceConnected method is called when a connection to the Service has been established, with the IBinder of the communication channel to the Service.It takes in two parameters as input.First is the ComponentName which is the component name of the service that has been connected. Second is the IBinder which is the IBinder of the Service's communication channel. The onServiceDisconnected API is called when a connection to the Service has been lost. It takes a single parameter as input namely the component name which is the component name of the service whose connection has been lost.

IV. IMPLEMENTATION

We implement the Video streaming application over the PEC service with the help of the API provided by the PEC service.For the implementation of this application we connected 2 smart phones to a hotspot created by using a third mobile.The smart phone that we used had Android version 6 and smart phone via which the hotspot was created had android version 4.The following flow chart explains the sequence of operation one has to perform in order to stream videos or share files.

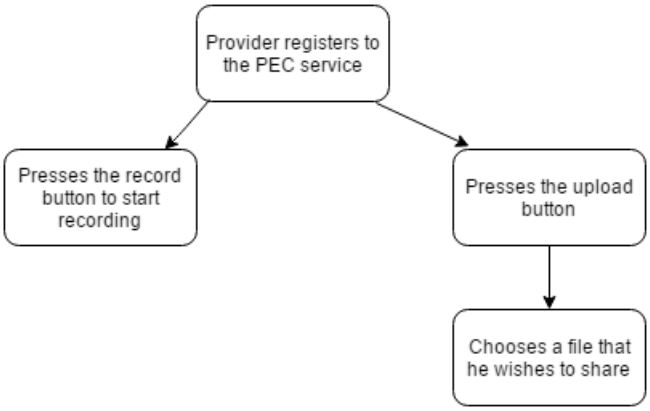

Fig. 2. Flow chart sequence operation that the provider follows to share data

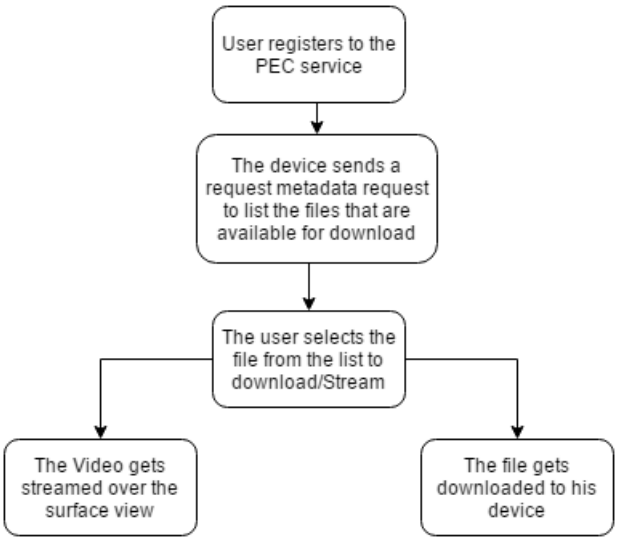

Fig. 3. Flow chart sequence operation that the user follows to receive data

As shown in Figure 2 The provider first registers to the PEC service.Then if he wishes to stream a video then he presses the Start recording button. The streamed videos are then transmitted to the requesting device. If the provider wishes to share a file, he presses the upload button. This will open up the file explorer on his device. The provider then selects the file he wishes to share. The selected file is then converted to a byte stream and is transmitted to the requesting device via the PEC service.

As shown in Figure 3,the receiver first registers to the PEC service.Then the device sends a request to metadata API which fetches the list of available videos and files that are currently available. These files and location of the stream are then displayed in the user's device.The user then selects any of the available images that he wishes to download or the stream that he wishes to view.The video byte stream are then provided by the service which gets rendered on the Surfaceview for display.If it was a file that he wants to download , the byte stream of the file contents are provided by the service which is then saved on the users device.

V. EVALUATION

The nature of any android application is such that there may be a possibility that the expected output is not received since there are various factors involved which decides the correct output like the android version, the compatibility of the devices sharing data among themselves, the minimum sdk required etc. Since it is a field which keeps getting updated with time, the devices that are little older dont accept the new codes, while old codes may or may not work in the new devices. This poses a serious challenge in making an application that is compatible with the devices, as well as gives the correct output.

In our evaluation, we have tested with many different values of the parameters in order to get the best results. Like, the minimum sdk version that we have used is 19. Whereas when we tried working with minimum sdk value as 23, we were not getting good results. We tried with different minimum sdk value for different android versions. We found out that for the android version of 6.0.1 (Android Marshmallow), the value of 19 for minimum sdk version gave out the best results. The data transfer speed was quite high for the file size of less than 8 Kb.

For evaluation and testing part of our system, we used two smartphones with same android version installed in both the mobile devices (android version 6.0.1), and a third mobile phone was used as a hotspot to which the other two mobile phones were connected. In this way, we were able to connect both the mobile phones in the same network.

VI. DISCUSSION

As we mentioned in the beginning of the paper that we are building a system that would require improvements as and when the software updates take place, since it has been built using the most important tool called Android Studio, and we know that android undergoes many software updates. Due to this, we have faced various challenges in building up this system. There have been cases where one part of the code would be working in a lower version of the android system but same was not the case when trying in the higher versions of it. Due to this, we had to make some compromises which might have lower down the optimization levels which could have been achieved if we wouldn't have faced with such issue. This can be attributed to one of the limitation of the system. Also, as we have stated that we have built an application that is integrated with the PEC Service, we have found some of the hidden flaws or bugs in the service. These bugs are needed to be properly addressed before making this system a complete success. The rate of data transfer is still quite low and due to this, we were not able to sufficiently transfer large bytes of data and stream it over mobile devices. In the case of file sharing, we were able to transfer files that are of smaller size (in 5-8 Kb range) from one device to another, but trying with files larger than this size resulted in longer waiting periods and still not achieving the desired goals. Since the rate of data transfer is quite slow at the moment, we are unable to stream the videos perfectly on any of the device. We even tried to divide the large data into smaller chunks, as we have proposed in our potential solution to the challenges section, but there are still some constraints present in the PEC Service, which is stopping this to work. We need to fix these bugs before any significant improvements be made in case of transferring larger data.

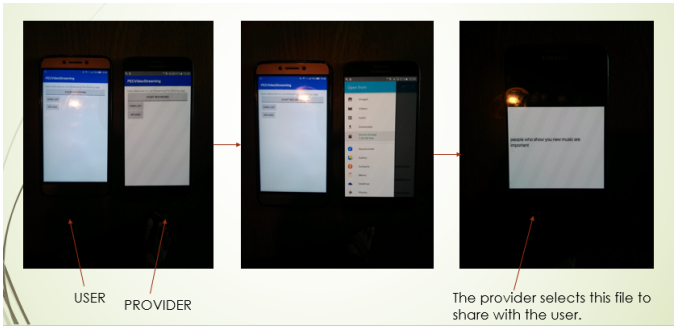

Fig. 4. Sequence of operation performed by the provider to share the file

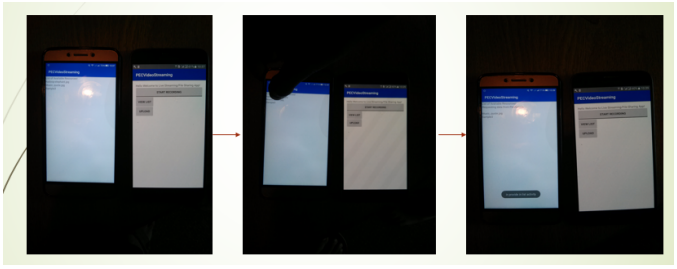

Fig. 5. Sequence of operation performed by the user to download the file

But we can see that if these minor changes are placed in effect, this system has the potential to solve a huge problem of sharing meaningful data among a group of users so that everyone around the system are made aware of their surroundings.

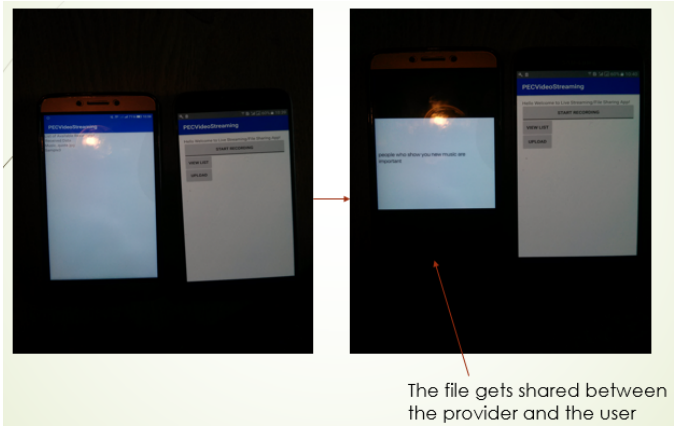

Fig. 6. Steps showing the working of PDS: File Sharing

Currently any transmission of data with less than 10 Kilobytes is possible and the system behaves in exactly the same way as it is supposed to work in. In case of such size of data, the speed of transfer is also quite fast, making the application work with a very high probability (that the packets of data are not lost in between). If there is a way that the larger data, once converted into chunks of smaller data, can be transferred among mobile devices, this problem of larger data being not able to transfer, can be easily resolved.

With the above problems being taken care of, the future plan of action may include filtering of the videos based on the popularity and relevance of the user, once he signs in (similar to what we might have seen in case of YouTube). This would further optimize our existing system. Also we could design our system in which, depending upon the bandwidth available, we could stream the videos with different quality options. The limitation of access points can also be addressed and the PEC Service can be further improved so that it enables data sharing even if the user as well as the provider are connected in different network. further, the application can be modified to support voice communication and video chatting like the applications like Skype support. All these ideas can be clubbed into one single application, thus making it one of the most unique application present today.

VII. CONCLUSION

We have designed a small prototype to show how data can be shared among peers without worrying about the internet connection. This is just to show how effective such kind of a system can become if all the challenges addressed are suitably handled. Currently our application is capable of sharing files in different formats among mobile devices, whose size is limited to a certain range. This application is built over PEC Service which acts as a binder and enables data sharing when the provider as well as the user are connected in the same network. We have proposed a system that extricates the boundation we normally have, of depending upon the internet connection in order to share the useful data among opportunistically gathered devices. Another advantage of this fully functioning application would be that this would provide on-the-go streaming of the video feed, thus providing live details. Thus, we have proposed an application built over peer data sharing service that enables mobile devices to discover and retrieve data among peers, which is valuable for many daily scenarios where high densities of mobile devices congregate opportunistically.

REFERENCES

[1] Robust, Efficient Peer Data Discovery and Retrieval among Mobile Devices
[2] www.stackoverflow.com
[3] www.tutorialspoint.com
[4] developer.android.com