Extension of the PingER Project onto Mobile Devices using Android Applications

Aayush Jain [1]
Amity University
Noida, UP, India
aayush.2896@gmail.com

James David [2]
Amity University
Noida, UP, India
jamesdavid.1997@gmail.com

Rishabh Bansal [3]
Iowa State University
Ames, Iowa, USA
rbansal@iastate.edu

Prof(Dr) A. Sai Sabitha [4]
Amity University
Noida, UP, India
assabitha@amity.edu

Prof(Dr) Abhay Bansal [5]
Amity University
Noida, UP, India
abansal1@amity.edu

Dr Les Cottrell [6]
SLAC National Accelerator Laboratory
Stanford, CA, USA
cottrell@slac.stanford.edu

Prof(Dr) Bebo White [7]
SLAC National Accelerator Laboratory
Stanford, CA, USA
bebo@slac.stanford.edu

***Abstract* – PingER was developed by the SLAC National Accelerator Laboratory (SLAC) as a tool for Internet End-to-end Performance Monitoring. It monitors over 700 sites worldwide, and aims to measure the round-trip time, loss jitter etc. for packets travelling between nodes on the internet. The PingER MeasurementAgent can be deployed on servers running Linux, however these servers have limitations. The fixed-line servers currently in use are not mobile and require a continuous power source. The extension of the PingER project to the Android ecosystem brings advantages like greater power efficiency, ease of installation, maintenance, and better affordability to the table. The Android application supplements the Measurement Agents set up at about 40 locations around the globe.**

Keywords – PingER, Android, FTP, RegEx, Automation, Asynchronous Computing

Introduction

The SLAC National Accelerator Laboratory's (SLAC's) Internet End-to-End Performance Measurement (IEPM) Group backed by the US Department of Energy aims to gain valuable insights into the performance of the Internet. This research-implementation targets replicating the entire PingER workflow into a portable Android application. This allows Android devices to act as remote nodes to any specified measuring agent which ultimately allow for internet end-to-end monitoring. The Android app acts as a Measurement Agent (MA), sending out pings to a SLAC-hosted list of beacons every half hour, and recording their responses. This data is saved and sent to the PingER archive at SLAC for use in multiple projects.

When it was started in 1995, the primary goal of PingER was to "keep tabs on how parts of the network were performing and root out any problems" so as to know how the Internet was performing, identify problems, and apply solutions. Now, it has expanded to something bigger – identify and assess the 'digital divide' across different regions of the world from Sub-Saharan Africa to the Middle East, from South America to Central and South Asia. This 'digital divide' refers to economic and social disparity with regard to access to information and communication technology. The project has various subdivisions such as PingER Deployment, Analysis, Operations, Databases, Validation data and toolbox that further open up multiple avenues like informed decision making.

Literature Survey

PingER:

PingER is a Project led by the SLAC National Accelerator Laboratory and developed by the IEPM group in 1995. It is short for Ping End-to-End Reporting. The framework for the PingER project is based on the ping utility, that is available on most Internet connected hosts. A ping involves sending an Internet Control Message Protocol (ICMP) echo request to a specified remote/target node which responds with an ICMP echo reply. It is also optional to send a data payload in the request which will be returned in the reply. The round-trip time (RTT) is reported; if multiple pings are dispatched, most implementations provide statistical summaries. PingER uses this data to assess the quality of the internet in various regions, understand performance, and identify problems. For each remote node specified in a configuration file:

- PingER sends a single ping with a 56-byte payload,
- followed by up to 30 pings with 100-byte payloads at 1 second intervals to the remote node, until 10 response are received.
- This is followed by sending up to 30 pings with a 1000-byte payload, also at 1 second intervals to the remote node until 10 responses are received.

Android:

Android is a mobile operating system developed by Google. It runs on a modified version of Linux adapted primarily for touchscreen mobile devices. Android's source code has been made open source by Google, a member of the Open Handset Alliance. It is used on more than 2 billion active devices worldwide by more than 1 billion users and has a huge community support. Android enables developers to create compelling mobile applications that leverage the modern capabilities a handset has to offer, so as to create richer and more cohesive experiences for users. This was one of the main reasons why Android was the platform of choice for the implementation of this project to create a robust Java-based application.

RegEx:

A Regular Expression (RegEx) allows a programmer or a user to define how a computer should search for a certain string in lots of text. It helps match, locate, and manage text while utilizing advanced pattern matching. A majority of programming languages provide native support for RegEx via standard libraries, which can further be expanded by using extended libraries. Regular Expressions mean the specific, standard textual syntax for pattern representation for text to conform to. RegEx uses 'metacharacters' to define the search term. A text-directed RegEx engine walks through the subject string, attempting all permutations of the regex before advancing to the next character in the string.

FTP:

The File Transfer Protocol (FTP) is a standard network protocol that is used for the transfer and storage and retrieval of files between computers over a TCP/IP connection. It utilizes the client-server architecture. It allows authenticated, as well as anonymous connections. For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS) or replaced with SSH File Transfer Protocol (SFTP). The project utilizes the FTP Protocol to send the PingER output generated by the apps in the Android MA hosts to a proxy server at a PingER archive site, such as the one at Amity.

Existing Model

The project as implemented by Rajappa, Sampson et al involved running the Perl scripts as supplied by SLAC on a rooted Android Device. This was a relatively low-effort and high-maintenance approach to test out the feasibility of Pinger on Android.

The model was found to be largely unsuccessful because of the intensive effort required to set it up in the first step. For starters, the proposed and built application would only function on rooted devices, and the majority of the Android Devices are not rooted, i.e., they do not have SuperUser Access to the device. Freshly purchased devices do not provide root access to the users, and users need to flash their device to gain root access. This is not feasible for a majority of users as their mobile devices are not rooted by default.

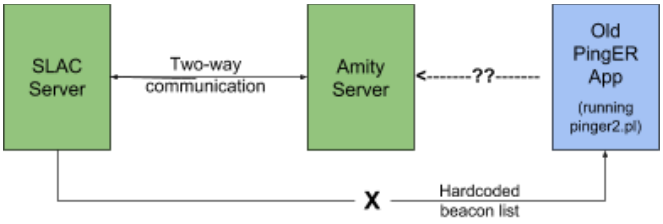

Figure 1 Graphical Representation of Old Model

The model employed the following approach:

1. Root Android Device, and gain access to system commands and directories.
2. Install BusyBox and any Terminal Emulator app.
3. The Perl Scripts as supplied by SLAC were made to run in the terminal (in the background) with a frontend of a Java App UI which would run the required commands on the appropriate user inputs.
4. The application had the beacon list hard-coded, and thus auto-updating it was not possible. Only the beacons as hard-coded would get pinged.
5. The application was not set to auto-start on device power on. Even the pinging function had to be done manually.
6. The output log generated in txt were available to un-rooted users too on external storage, making it susceptible to tampering.

Implemented Model

The proposed model revolves around the implementation of data collection through PingER solely over Android, which replicates the earlier model of Linux executing the pinger.pl Perl Scripts as supplied by SLAC. A prototype was developed for the existing model which wasn't efficiently feasible and had certain drawbacks and inefficiencies. We've proposed and implemented a model which sports much more accessibility and has other features too.

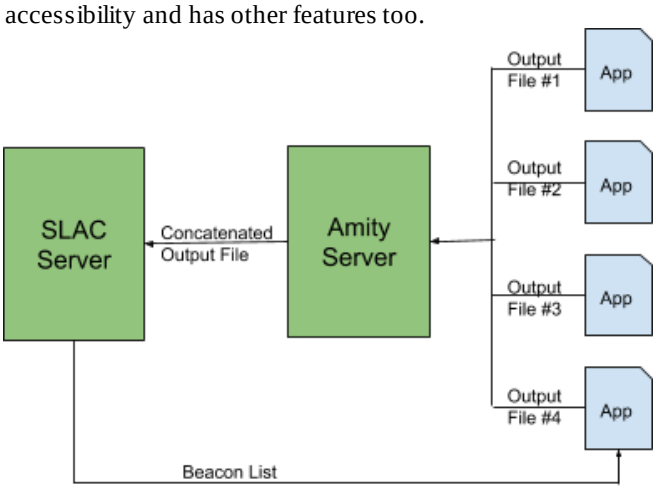

Figure 2 Graphical Representation of Newer Model

To overcome the shortcomings of the previous model, a new modular application was written from scratch in Java. The new model also runs on non-rooted Android devices as well. After booting the Android MA (measuring agent), the Android PingER App is programmed to automatically launch in the background on device startup. The app continues to run unless the Android MA experiences a power failure, or when the MA is turned off for some unknown reason, or when the system restarts with a deleted cache, regardless of the error. This service will keep on executing even when the application is closed. This has been done to ensure a procedure does not fail to be invoked. The data generated by the application is transferred to the Server hosted at Amity University MA at a certain time interval (every day at 9am IST) using the FTP protocol.

The new model employs the following approach:

1. No root is needed for proper functioning of the application.

2. On Device boot, the application launches itself in the background to run services like updating beacon lists, and pinging them.
3. The beacon list is now dynamically updated automatically from the list hosted by SLAC. The update occurs at least once a week, and this list is synced locally.
4. This local beacon list is pinged automatically once a day. Manual pings can be done as and when needed.
5. The output from the ping commands is parsed using RegEx and written to a date-wise file.
6. This generated txt file has now been shifted to the App's dedicated internal storage that cannot be tampered by any user.
7. A new txt file is generated every day, and outputs for each day are appended only to that day's txt.
8. FTP has been chosen as the protocol of choice for the transfer of generated txt files from the Android App to the MA Amity Server acting as a proxy between the app and SLAC.
9. Files that have been created and not transferred over FTP are kept track of in the App so that upon the next sync, only un-synced files are uploaded. This also allows us to manage older files which can now be safely deleted so as to use space conservatively.

Methodology

Subsequent to the boot up of the Android MA, the Android Application PingER is programmed to run automatically in the background and keeps running unless the Android MA suffers from a Power Failure. If the MA shuts down due to an unknown reason during any moment of any process, it will be restarted irrespective of the error with a cleared Cache. This service runs even when the application is closed. This was done to ensure a procedure i.e. the data being generated by the application is transferred using the FTP Protocol to the SLAC anonymous inbound FTP server at a specific time interval. Thus, at every iteration of the time interval per week, the generated pinged data is allowed to be pushed into the SLAC FTP server automatically.

In the previously proposed models, there were several concurrently running threads which increased the Android workload and CPU utilization; and thus, resulted in a delay in fetching the Beacon List. This was a major drawback in the older versions of Android phones with fewer cores. To overcome this, we have been utilizing only two services which are running consecutively: one, with instructions containing methods to check the internet data availability; and the other, an asynchronous task involving two other methods. These methods establish the connection, download the pinger.xml configuration file and parse through it to extract the beacon list. Previously, this functionality of Android PingER was implemented for a small number of beacons which were hardcoded.

Prerequisites (A-Synchronised Instructions):

There may be a successful internet data connection which the device is accessible to, If the internet data exchange is not applicable or not working due to reasons such as No Network or No Data Connection Availability, then the application uses the previously parsed pinger.xml configuration file.

A-Synchronous Task:

Prior to parsing the pinger.xml file to extract the beacons, we need to download the file itself from the SLAC URL (http://www-iepm.slac.stanford.edu/pinger/pinger.xml).
The application gets access permissions to the Android root directory and checks if there exists a folder with the name PingER, if not found then it creates a new directory with name PingER.
The task of downloading the pinger.xml file containing the TAGS for the beacons with their IP names and addresses is then executed. Then the downloaded XML file presently in the buffer is appended to a file locally. This task of downloading the XML is executed every week to ensure an up to date list.

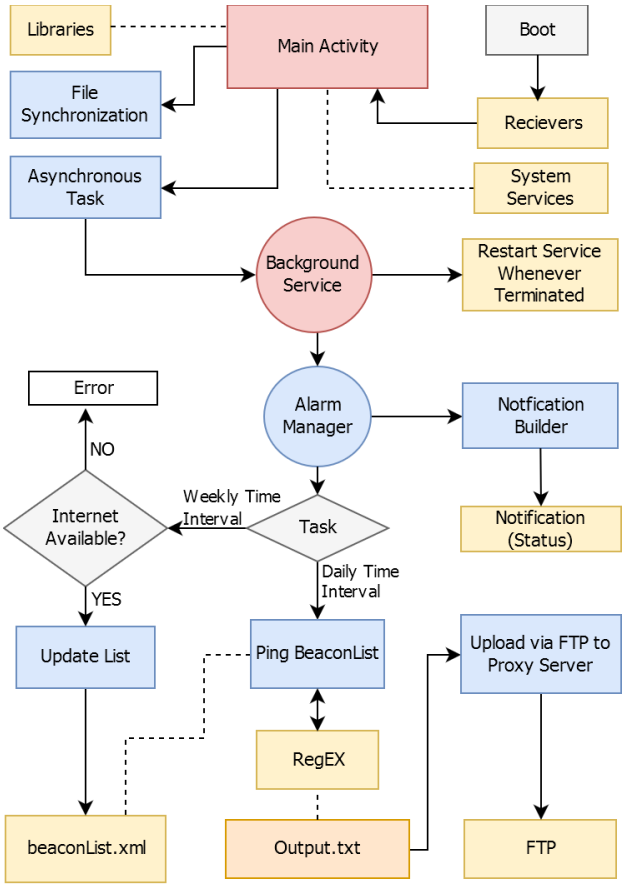

Figure 3 Automated Workflow for Android Application

Step 1: Collection/Pinging

Collection: Upon the successful download of the pinger.xml file, it goes through the parsing phase where the information regarding the beacons such as Name, IP address are extracted. This information is necessary for further processing phase i.e. Pinging. The parser we are using is SAX Parser. This is very efficient for Android as it does not load the complete document into memory, rather it takes statements into buffer and checks for tags such as starting tags and ending tags to get the required the data from the xml and

discarding other tags. Additionally, the beacon list can be refreshed manually using the swipe-to-refresh feature.

Pinging: The collected beacon hosts are passed through the ping command: **pingCmd = "ping -n -c 10 -w 30 -i 1 -s 100" + host**; where c is number of pings it awaits a response from, w is the time to wait for a response in seconds,, i is the interval between pings, and s is the packet size for the pings.

Table 1 Parameters recorded by Android Application

S.No	Parameter	Description
1	Monitor_Host_Name	Name of the host
2	Monitor_Addr	IP address of the host
3	Remote__Name	Name of remote site
4	Remote_Addr	IP address of the remote site
5	Bytes	Bytes of data sent
6	Time	Round Trip Time
7	Xmt	Number of packets sent
8	Rcv	Number of packets received
9	Min	Minimum response time for packets in ms
10	Avg	Average response time for packets in ms
11	Max	Maximum response time for packets in ms
[12..(12+Rcv)]	Seq[1..Rcv]	Sequence numbers of individual responding pings
[(13+Rcv)..(13+2xRcv)]	RTT[1..Rcv]	Round trip times of individual responding pings

Step 2: Parsing, Storing

The host name for the generating outputs is "pinger-and1.amity.edu". Certain tags such as hostIP, Time Stamp, Packets sent, Packets Received are involved in the output. The outputs generated from the ping command were fetched using RegEx pattern matching and stored in a file locally currently named as data.txt in the MA. If the file already exists for that particular iteration of time in a specific week, then the latest generations will be appended to the existing file or else a new file will be created and data will be appended into the same.

The functions used in RegEx to parse and format the output are:

Table 2 Functions used for Parsing in Android Application

Function	Pattern	Purpose
getHostIp()	None	This function is used to fetch the IP of the host using the inbuilt functions of Android using the WiFi manager class
parse-Group-Timestamp()	"\\[([0-9]{10})\\] PING"	This function searches for the Unix timestamp for the ping command within the group-Timestamp class
parse-GroupIP()	"\\(([0-9,\\.]+)\\)"	This function is used to get the IP address of the group that is being pinged.
parse-Group-URL()	"PING\\s+([\\w,\\.]+)\\s+\\("	This function parses the entire URL of the group and returns a string containing the host that is being pinged.
parse-Group-Bytes()	None	This function determines the number of bytes that were sent. It is done using a two-stage pattern comparison to get the string
getNumberOfPacketsSent()	"\\d+ packets"	This function is relatively straightforward and finds the number of packets that were sent using the pattern
getNumberOfPacketsReceived()	"\\d+ received"	This function acts similarly as the above method to collect the packets received
countICMP()	"icmp_seq=\\d+"	This function is used to determine the number of ICMP sequences sent
timeOFEachIcmp()	"time=\\d+"	This returns a string value of the time of the ICMP sequence
parsePingStatistics-MinAvgMaxMdev()	"ping\\s+statistics\\s+[\\-]+\\[([0-9]{10})\\]"	This function extracts the max, min, average and the Mdev for the ping command and returns a string array for each Min, Max, Avg and Mdev respectively.

The query used to ping the beacons is: **pingCmd = "ping -n -c 10 -w 30 -i 1 -s 100" + host**.

The output of the ping command, viz. the data received from pinging the beacons is of the form:

Table 3 Data Written to Output File after Regex Processing

pinger-and1.amity.edu 192.168.0.100 www.andi.dz 213.179.181.44 100 1537559939 10 10 193.975 207.584 255.100 1 2 3 4 5 6 7 8 9 10 198 255 202 207 209 202 200 193 201 204
pinger-and1.amity.edu 192.168.0.100 linhost2.utic.net.ba 195.130.35.104 100 1537560843 10 10 184.845 187.625 190.790 1 2 3 4 5 6 7 8 9 10 184 186 187 187 186 187 190 187 188 189
pinger-and1.amity.edu 192.168.0.100 megalan.bg 95.111.55.250 100 1537560853 10 10 184.566 190.618 193.917 1 2 3 4 5 6 7 8 9 10 184 188 193 188 191 190 190 193 191 191

```
pinger-and1.amity.edu 192.168.0.100
www.carnet.hr 161.53.160.25 100 1537560862 10
10 164.157 173.023 175.944 1 2 3 4 5 6 7 8 9 10
170 164 175 174 173 173 175 173 173 173
```

Step 3: Send to Server

The FTP protocol is being used to transfer the data from the Android agent to the Amity Proxy Server which enables the load generated by the web-server to be differentiated into three fragments:

1. FTP Client Authentication: The device requesting to acquire the service of the FTP is authenticated anonymously, tested in a local server built up with address "ftp://192.168.0.101" and port number 2121. If the client does not authenticate or has any issue regarding the connectivity to the internet, then the daily upload of the file is rescheduled and is set to upload on the next day along with the generated data.txt for it.
2. Check Current Repository: The application segregates the generated data.txt files for the current date from the rest and checks if there are other files which weren't able to be uploaded to the Server. All the files to be uploaded are flagged prior to the process.
3. FTP Upload To Server: Finally, the files are uploaded to the server if the above steps are executed successfully, else the application will wait for the next day for the AlarmManager Service to generate an alarm which would enable the FTP service to be executed once again. If the upload process of the files is successful, then they are set to delete automatically. If the process is unsuccessful, then the batch of uploaded files will be transferred in the next FTP service. Once the batch has been uploaded it will be deleted.

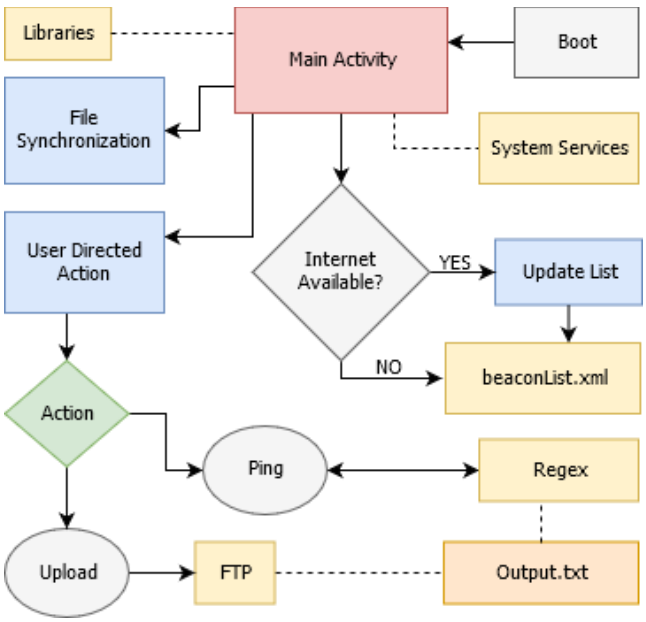

Figure 4 Manual Workflow for Android Application

Miscellaneous:

Subsequent to the Device Booting, a notification service is processed which gives the controls to AlarmManager (AM) service. The AM service sets up an alarm for daily uploads and weekly beacon list updates.

All the data is being stored in device system storage. The application folder is located at: /data/data/"your_package_name".

This folder can be accessed only using the DDMS for your Emulator, Since this can't be accessed on a real device unless you have rooted it, it would promote security to the profiles generated and won't be feasibly available to illegal use.

Conclusion

The porting of the PingER application onto Android-based mobile devices has been a successful project. The targets set for the application such as ease-of-access, maintainable code, scalable architecture, small foot-print and viable scalability have been achieved. This latest iteration of the project overcomes the hurdles of the previous iteration of PingER on Android wherein the Perl Scripts were run on an emulator on a rooted Android Device. The Android App can now dynamically ping the beacons as and when updated by SLAC, parse and store their output, and send the generated txt files on a daily basis to the server via FTP – all in the background requiring minimal user-intervention. This approach allows individuals with minimal technical knowledge to contribute to the PingER Project with little to no effort which will be a huge boon to the data collection team at SLAC. This, in turn will be a greater boon to the data analysis team as the number of android-based measuring agents increase and provide more diverse data. This app can now be distributed to people around the world for collecting data.

On wired measuring agents, i.e. servers with a dedicated fixed-line internet connection, there may not be much variation in the quality of data recorded over time. However, when the PingER Project is expanded to wireless Measuring Agents (Mas) such as Android Devices, which rely either on either mobile data or WiFi for Internet connectivity, the results are sure to be interesting because of the diversity wireless measuring agents bring to the table.

The performance improvements recorded in this new iteration of the Android application have been exemplary. And there is still more to achieve. The possibilities arisen by porting PingER to Android are numerous. This allows the project maintainers to either run the Android Application in conjunction with the existing project as an extension; or as a separate branch parallelly after some modifications as needed.

Future Proposals

The Extension of the PingER Project onto Mobile Devices using Android has opened up a world of opportunities for adding up new measuring agents at the click of a button. A one-click installation procedure now enables SLAC researchers around the world to start pinging beacons, recording the output and conveying the logs to the proxy

server via FTP with just one button press, something which has also been automated to run as a service in the background thereby requiring minimal or no human effort now.
Currently there are two possible routes for future expansion of the project:

1. Extend the capability of the current application: This method requires modification of the pinger2.pl script so as to spin up a dedicated FTP Service too. This allows the existing hosts to open up their ports to receive incoming files from various deployed Android based MAs. On the client side, i.e. the Android App, the user can be presented with a choice to select which ArchiverMA they want to associated with during signup; so as to send the generated txts to a selected Archiver. This method has the only risk for any bad actor to exploit the open FTP servers in case authentication is not used. To overcome this issue, the pinger2.pl script needs to add a file monitoring service too, which blocks all incoming files that are not in txt format. Because open FTP servers can act as a harbor for illegal files in formats ranging from exes, jpegs to even mp4s. The monitoring service can parse through received text files to see if they are in a pinger-compliant format; and the rest can be flagged or simply discarded. Another possible drawback is that individual hosts cannot be identified to track which file was sent by whom. To overcome that SSH or other authentication can be used, but that again is a trade-off with ease of access. Non-password-based authentication may not feasible because adding and removing public SSH keys onto the keyrings must be done manually or in another secure fashion. Our best bet in that case must develop a proper REST API, and forego the FTP Services and ease-of-access as well. This can thus act as a more centralized model wherein the server lies with SLAC, and every other measuring agent or application in the world sends data to SLAC servers in US. This brings us to our other possible route for future expansion.

2. Firebase Application: The need of a proxy server can be completely eliminated by shifting to a cloud-based architecture for managing files. Instead of SLAC pulling in the generated txt files from MAs around the world, the MAs can themselves push these files to a centralized application server hosted on the cloud; which makes it easier for SLAC to access files as per their need. Considering the great degree of integration capabilities Google offers with its products, the flexibility arising from using the Google Cloud Platform would be pronounced. Google's Firebase is a mobile and web application development platform that provides developers with a variety of tools and services to help develop high-quality apps, that can scale easily as per changing demands, and which delivers 99.99% uptime. The Firebase SDK allows mobile app developers to quickly add critical and reliable functionality to their applications in a short time. The recommended option here would be to leverage the Cloud Storage for Firebase that allows robust uploads and downloads onto the Google Cloud Storage buckets. Apart from the user authentication module that comes bundled with Firebase, developers can also declare file security parameters so as to allow only certain filetypes to be uploaded. Having all user uploaded files in one place will then enable SLAC to access and process files as and when needed. Server-side processing can be done on the Google Cloud Platform as well, thereby eliminating any need for SLAC to maintain its own physical infrastructure. This form of implementation can be highly beneficial in any kind of region of the world, also including remote places, like deep inside tropical rainforests, or places that have recently been hit by a natural calamity. As this implementation model is not dependent on any local measuring agent, the android mobile apps can directly deliver data to SLAC over any form on internet connection in minimal time with high reliability.

Although these suggested models have a grand vision for execution of SLAC's PingER Project, they are viable nonetheless. Moving over from legacy code and concepts might actually provide a huge boost to the variety of data being collected for IEPM by SLAC's PingER Project. Nevertheless, this accomplished implementation is good enough to run in conjunction with the present project so as not to lose any form of functionality; and at the same time not making a trade-off with either functionality or ease-of-access.

References

[1] W. Matthews and L. Cottrell, "The PingER project: active Internet performance monitoring for the HENP community," *IEEE Communications Magazine*, vol. 38, no. 5, pp. 130-136, May 2000.

[2] L. Cottrell and W. Matthews, "Measuring the digital divide with PingER," *SLAC*, no. SLAC-PUB-10186, p. 4, October 2003.

[3] S. Ali, L. R. Cottrell and A. Nveed, "PingER Malaysia-Internet Performance Measuring Project: A Case Study," 2015.

[4] S. M. Khan, L. R. Cottrell, U. Kalim and A. Ali, "Quantifying the Digital Divide: A Scientific Overview of Network Connectivity and Grid Infrastructure in South Asian Countries," in *16th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2007)*, Victoria, Canada, 2007.

[5] L. Cottrell, M. Warren and C. Logg, "Tutorial on internet monitoring & PingER at SLAC," 2000. [Online]. Available: http://www.slac.stanford.edu/comp/net/wan-mon/tutorial.html.

[6] L. Cottrell, "How Bad Is Africa's Internet?," *IEEE Spectrum*, 29 January 2013.

[7] L. Cottrell, C. Logg and J. Williams, "PingER History and Methodology," in *2003 Round Table on Developing Countries Access to Scientific Knowledge October 23-24, 2003*, Trieste, Italy, 2003.

[8] R. Sampson, S. Rajappa, A. . S. Sabitha, A. Bansal, B. White and L. Cottrell, "Implementation of PingER on Android," in *7th International Conference on Cloud Computing, Data Science & Engineering - Confluence*, Amity University, Noida, India, 2017.

[9] W. Matthews, C. Granieri and L. Cottrell, "International network connectivity and performance, the challenge from high-energy physics," no. SLAC-PUB-8382, March 2000.

[10] L. Cottrell, "Proxy support for PingER - Internet End-to-End Performance Monitoring - SLAC Confluence," 2018 September 2018. [Online]. Available: https://confluence.slac.stanford.edu/display/IEPM/Proxy+support+for+PingER.