INNOVATIONS IN MOOC PLATFORMS FOR IMPROVED USER EXPERIENCE

Aditya Vadlamani, Revathy KT, T V Prabhakar
Department of Computer Science, IIT Kanpur, UP 208016, India
{adityav, revathy, tvp}@iitk.ac.in

Abstract

Massive Open Online Courses or MOOCs quickly established themselves as important components of education. The software for delivering MOOCs is a MOOC management system - a well-known example being openEdX. A MOOC management system is a sophisticated piece of software and is typically designed to deliver MOOCs for an audience with good internet connections. Often this is not the case in a developing country - the bandwidth is limited, costly, and intermittent. With this in mind, we designed and developed a MOOC Management System, mooKIT, with innovative ways of addressing this issue. We also address the problem of students being dependent on the credential issuing authority for verification.

Push/Pull & Data Caching: The traditional pull model between client and server is replaced with push models using messaging protocols, making it similar to WhatsApp. The data, including videos, can be cached at the client and need to be accessed only once. This reduces the bandwidth requirement substantially.

Semi-Online mooKIT: Static content in MOOCs like videos take up a substantial volume, normally in gigabytes. Whereas dynamic content like Announcements takes up less space. Static content can be distributed using an SD Card, and an Internet connection can be used to get the dynamic content. This delivery method facilitates the user to have a full-fledged MOOC experience without the need and expense of a high bandwidth internet connection.

Offline mooKIT: Archived courses on mooKIT are bundled into a mobile app. All the course information can be accessed locally, thus giving a complete MOOC experience without an internet connection.

Digital Certificates: Cryptographically signed digital certificates are issued on public blockchains like Bitcoin. They are tamper-proof and independently verifiable. This removes the dependency on the issuer for verification, giving complete ownership of the credentials to the students.

1. INTRODUCTION

MOOCs give the flexibility to learn many things from any place with an internet connection from the best of teachers. Right from finding jobs in the tech industry to augmenting college-level education in diversified fields like agriculture, arts, education, sciences, etc., they have quickly become an indispensable means of high-quality learning material.

With more and more people using mobile phones in developing nations, the opportunity to deliver educational content to the masses has never been better. However, addressing the limited and expensive bandwidth is still a challenge in many developing nations. To make online learning accessible to a broader range of audience, we have come up with innovations ranging from adapting more efficient communication protocols to delivering courses without good internet connections.

Blockchains provide immutability, transparency, and verifiability to data records. They have been proposed to be used in everything, ranging from supply chains to government records. mooKIT[1] uses digital certificates issued on blockchains to address challenges with certifications in MOOCs like tampering and dependency on issuers.

In this paper, we discuss these details.

1.1 mooKIT

mooKIT is a lightweight MOOC management system, designed and developed at Indian Institute of Technology, Kanpur mainly for developing nations. It is designed based on the following principles:

- The platform must be easy and intuitive to use
- Course management must be simple
- Should not demand technical maturity from users
- Should be efficient with server resources
- Should be built using open-source technologies

1.2 Architecture of mooKIT

mooKIT follows a REST architectural style. It uses Drupal[2] content management system as the backend engine to handle the content and users. The main engine runs on MEAN stack, making it lightweight and extremely scalable with the ability to run on low-end servers. The architecture of mooKIT is shown in figure 1.

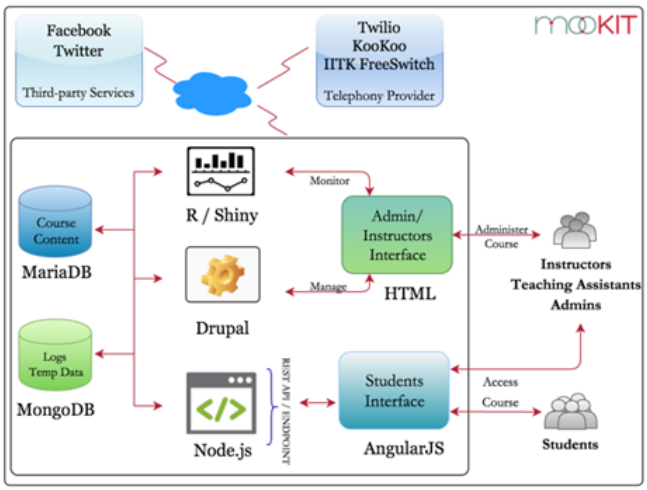

Fig 1: Architecture of mooKIT

1.3 Features of mooKIT

- Scalability: mooKIT is designed to scale by shifting most of the business logic to the client-side, thereby freeing the server resources. This architectural decision gives the server the ability to handle a large number of users at once.
- Analytics: mooKIT has a state-of-art analytics module, which provides a comprehensive analysis of student activities. It provides an easy to use analytics dashboard, which gives insights about the course as a whole and at an individual level.
- Interactions: mooKIT has integrations with social platforms like Facebook and Twitter as well as forums and hangouts on the platform where students can interact with each other. Using audio forums, users can express thoughts and ideas in detail rather than having to type.
- Certifications: Certificates for students can be automatically generated using templates based on the criteria set by instructors.
- Progressive mobile applications: mooKIT has mobile apps for iOS and Android platforms. The course content is cached locally, making it available even when the device is offline.

2. INNOVATIONS WITH PUSH/PULL MODELS & DATA CACHING

In many courses offered on mooKIT, students use mobile phones to access the course content. To make accessing courses easier, mooKIT has mobile applications for iOS and Android platforms. These applications would communicate with REST APIs using the HTTP protocol to obtain the latest information like new lectures, announcements, resources, forums, hangouts, etc. However, HTTP protocol consumes a lot of bandwidth due to network overheads. Laine and Säilä[3], compare the network overheads between HTTP polling and XMPP[4] over web sockets, which is shown in table 2.

	HTTP Polling	XMPP over web sockets
Ethernet Header	22 bytes	22 bytes
IPv4 Header	20 bytes	20 bytes
TCP Header	32 bytes	32 bytes
Protocol Header	365 bytes	2 bytes
Ethernet Footer	16 bytes	16 bytes

Table 2. Comparison of network overheads between HTTP and XMPP over web sockets.

Table 2 clearly shows that HTTP protocol headers are much higher than web socket headers. Besides, with every HTTP request, a new TCP connection is made, which can cause a lot of overhead between the mobile app and the server. Figure 2 illustrates the interaction between a client and the server using the HTTP protocol.

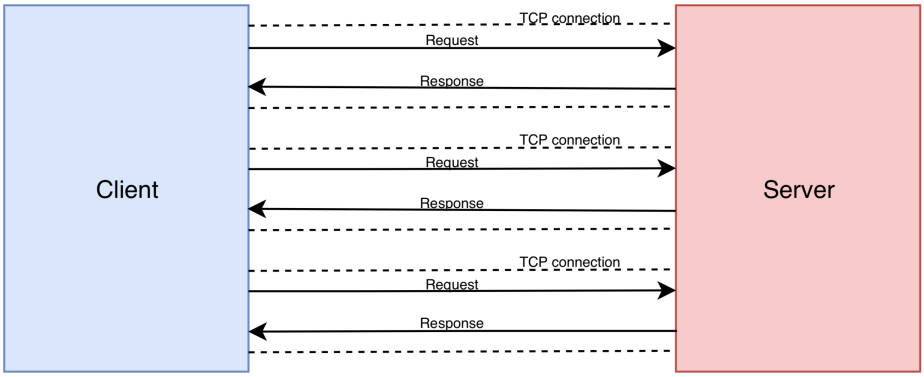

Figure 2. Interaction between client and server using HTTP

In low bandwidth scenarios, constant HTTP polling with large HTTP overheads causes delayed communication and can often result in pages failing to load. Another disadvantage with HTTP based application is, the client has to make a request every time to check for new resources. This constant polling results in wasted bandwidth when the server does not have new resources.

To address these challenges, a few changes have been made to the architecture. The HTTP protocol is replaced with XMPP protocol over web sockets. XMPP is a messaging protocol used in real-time communication systems and messaging applications. Unlike HTTP polling, which creates a new TCP connection with every request, XMPP connection over web sockets maintains a single connection throughout the session, thereby reducing the unnecessary network overheads. Figure 3 shows the interaction between a client and the server using XMPP over web sockets.

Since XMPP implements a push-based architecture, the new resources available at the server can be pushed to the clients, thereby making the latest information available to all users instantly while removing the necessity to constantly poll for new information.

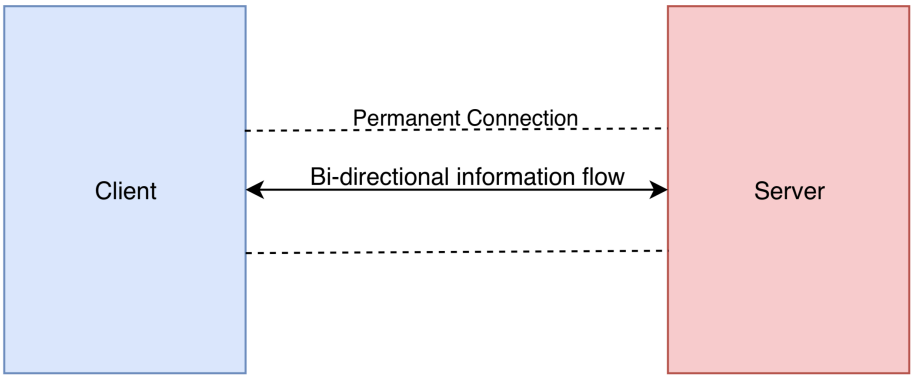

Figure 3. Interaction between client and server using XMPP over web sockets

2.1 Implementation

Figure 4 illustrates the modified architecture between the app and the server. As shown in figure 4, eJabberd[5] server, an open-source implementation of the XMPP protocol is added to the technology stack. eJabberd server acts as an intermediary between the NodeJS engine that runs APIs and the mobile application. It uses Mnesia DB to store user account information and content that has to be delivered.

2.1.1 Fetching the existing information

When the app is installed for the first time, a user account is created on the eJabberd server to uniquely identify each installation. The app then makes a request(I_{req}) to access all the course content like lectures, announcements, resources, assignments, and quizzes available until that point. eJabberd relays this requested information to NodeJS APIs, and the response(I_{res}) is returned to the app and is stored in the local SQLite database. User interfaces are rendered from the information stored in SQLite without having to fetch from the server every time, thereby saving bandwidth as well as making the renders fast.

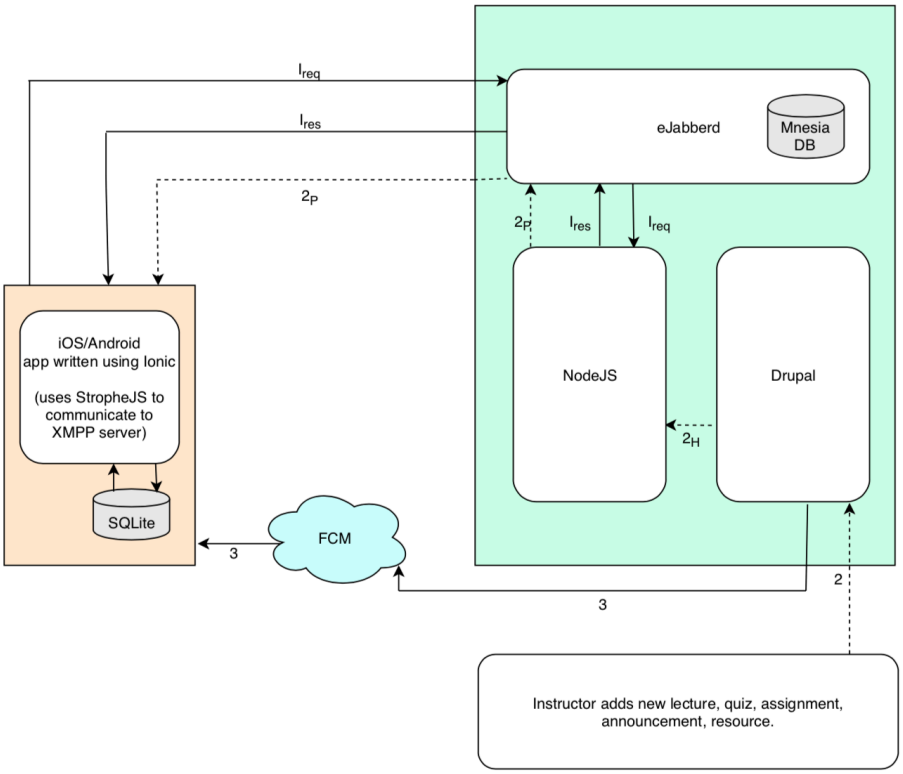

Figure 4. Interaction between mooKIT app and server using XMPP over web sockets

2.1.2 Pushing the new course content

Thereafter, whenever new content like lectures, announcements, resources assignments, and quizzes are added as shown in step 2 of figure 4 using Drupal, content is relayed to eJabberd using NodeJS APIs(2_H), which then pushes to all the available users(2_P). If the application is not active on a user's device, the content is stored in the Mnesia DB and then pushed when the app comes online. Push notification to inform the user about the new material is sent using Firebase Cloud Messaging[6]. This design decision helps in saving the battery as there is no need for separate processes running in the background all day. The newly obtained information is stored in the SQLite database along with existing information. This design removes the necessity to poll for the data, which saves the bandwidth.

2.1.3 Interactive content

The implementation varies slightly for interactive content like forums and hangouts. Since forums are dynamic, they are fetched whenever the user accesses the forums page and has an active internet connection.

Hangouts work similarly to any messaging application. When a user accesses the hangouts page in the app, the messages up until that point are pulled. When a new message is posted, the message is pushed to all the users who are active and are on the hangouts page. This information is not stored with eJabberd or in the local SQLite database.

2.1.4 Offline access

Since the information is stored locally, content like lectures, announcements, and resources can be accessed even if the app is offline. Videos can be downloaded to watch without an internet connection.

2.1.5 Video Analytics

As a user watches video lectures, the progress is tracked periodically and is stored in the local database. The stored video analytics are then posted to the server whenever an internet connection is available.

Table 3 contrasts the number of GET requests, which fetch the information from the server, between the app using HTTP protocol and the app using XMPP protocol over the web sockets when an internet connection is available.

	HTTP	XMPP over web sockets
Home page	Three requests on every visit	Three initial requests; new content is pushed
Announcements page	One request on every visit	One initial request; new content is pushed
Resources page	One request on every visit	One initial request; new content is pushed
Forums page	One request on every visit; One request for each post viewed	One request on every visit; One request for each post viewed
Hangouts page	One request to fetch the messages; Two requests every second thereafter	One request to fetch the messages; new messages are pushed
Profile page	Two requests on every visit	Two requests on every visit
Lecture page	Four requests on every lecture visit	Four requests on every lecture visit
Quiz page	Two requests on every visit	Two requests on every visit
Assignment page	Two requests on every visit	Two requests on every visit

Table 3. The number of requests made using HTTP protocol and XMPP over web sockets on each page visit

Comparison between the two protocols in the above table gives an idea of the network overheads that are avoided using XMPP protocol over web sockets. For example, if a student visits the home page in the HTTP app, ten times during a session, thirty requests are made to check for new content. On the contrary, in the XMPP app, three requests are made on the first visit, and whenever new content is added, it is pushed over the same connection that lasts over the session, without having to create a new TCP connection. Similarly, in Hangouts page, to emulate the real-time chat, two requests are made every second along with the initial request that fetches messages up until that point. If a user stays for two minutes in the Hangouts page, it amounts to 240 requests. Whereas in the XMPP app, the only initial request is made to fetch the older messages, and the new messages are pushed in real-time without the app having made one additional request.

3. SEMI-ONLINE mooKIT

Content type	Format	Size
Video files	480p resolution in mp4 format	2.4 GB
Resource files	PDF	94 MB
Lectures	JSON	17 kB
Announcements	JSON	11.5 kB
Resources	JSON	2.6 kB
Quiz	JSON	16.9 kB

Table 4. The data size of various course content in the MOOC Blockchains for Managers - Introduction and applications

Table 4 depicts the sizes of various course content in Blockchains for managers - Introduction and applications, a MOOC sponsored by COL and offered on mooKIT. Similar to the data presented in table 4, the primary source of bandwidth consumption in MOOCs is static content like videos, resources, etc., which can consume up to several hundreds of megabytes of bandwidth. Although watching video lectures is often a necessity to take a MOOC, this requirement can indeed be a burden to the students where bandwidth is expensive and limited.

Semi-online mooKIT approaches this problem differently. Rather than demanding a constant high bandwidth internet access, static content is served locally and requires one-time connection to download or can be obtained from peer groups.

3.1 Implementation
Semi-online mooKIT is a modification of mooKIT Android mobile application with XMPP over web sockets. It implements data push architectures, as discussed in section 2, thereby bringing bandwidth savings. However, it differs in the way static content is served.

3.1.1 Serving static content
In case of apps discussed in section 2, videos are streamed over the internet. Semi-online mooKIT apps render videos from the SD card or the internal memory of the android mobiles. When the app is run for the first time, it scans the internal storage and SD card for static content files, which can either be downloaded from the course website or be copied from anyone who has them(side-loaded). When the file path to the videos is obtained, video player switches to play videos locally. If any video files are missing, the app can stream videos over the internet similar to the completely online app. Resource files and additional resources to lectures are also stored locally and served, as they need high bandwidth compared to dynamic content.

3.1.2 Obtaining dynamic content
As discussed in section 2.1.1, Semi-online application fetches the dynamic content and stores in the local SQLite database, which is used to render the user-interface. New content is pushed to the app when the internet connection is available, as discussed in section 2.1.2. Similar to section 2.1.3, interactive content needs an internet connection.

3.1.3 Offline access and video analytics
Since the static content stays offline, the app can work without an internet connection and only needs a lightweight periodical connection to obtain the latest information. Similar to the discussion in section 2.1.5, video analytics are tracked periodically and synced with the server, when an internet connection is available.

4. OFFLINE-mooKIT

Offline mooKIT packages the archived courses offered on mooKIT into an SD card along with an android app. This solution does not require an internet connection.

4.1 Implementation
Once the courses are archived, the information about lectures, announcements, resources, and forums is collected in the JSON format and is embedded in the mobile application. The app then uses this information to render the user-interface. The static content, like videos and PDF resources, is packed into a zipped file, which can be downloaded from the offline mooKIT portal. Once the downloaded zip file is placed in the phone, the application scans the phone directories and finds out the path and unzips the file and is ready to play the videos. The progress is stored locally as the user watches the lectures. No login or user registration is required as there is no communication between the app and the server.

The course content can be shared among the peers from the app itself. The app provides the means to copy the course content from the internal storage to another SD card and vice-versa, what we call side-loading.

4.2 Potential use cases
Since an active internet connection is not needed, offline mooKIT has the potential to deliver OERs, archived, and reference courses to a broader audience. This is an excellent way to distribute the OERs which are MOOCs - they still appear as a ‘course’ and not just as a set of videos.

5. CERTIFICATES ON BLOCKCHAINS

Blockchains are a kind distributed ledgers which give you properties like immutability and non-repudiation - anything written on a blockchain cannot be erased, and the writer can not deny having written it. These properties are very useful in the case credentials issued to students, especially in MOOCs.

Certifications in MOOCs face challenges of their own. Students are from all over the world, and the certificates may be produced as credentials in a location which is far and foreign from the issuing authority. Blockchain-based certificates make it easy for anybody to verify the veracity of the certificate from anywhere in the world.

Typically, in MOOC management systems, certificates carry an embedded link which to the issuer’s website to verify the certificates. This model makes students dependent on the issuers and runs certain risks like the unavailability of the issuer’s website at the time of verification and the chance of malicious credentials if the website is hacked.

mooKIT uses certificates issued on blockchains to address these challenges using Blockcerts[7]. Blockcerts are a set of open standards and tools developed at MIT. Figure 5 describes various components in the mooKIT digital certificates.

5.1 Onboarding and Setting up mooKIT Wallet
Once a course is completed, invitations are sent to all the participants who successfully meet the criteria. This invitation has links to download mooKIT Wallet for iOS and Android platforms and one-time verification code.

Just like any cryptocurrency wallet, which can receive funds against a public address, mooKIT Wallet can generate public addresses against which the digital certificates are issued on Bitcoin blockchain. It implements BIP39[8], which aims to improve the user experience in generating and backing up the public addresses in the form of human-readable mnemonic words or passphrases. mooKIT Wallet uses Cert-Verifier[9], an open-source tool used to verify the digital certificates. It has provisions to share certificates across various social media platforms like Facebook, Twitter, WhatsApp, email, etc.

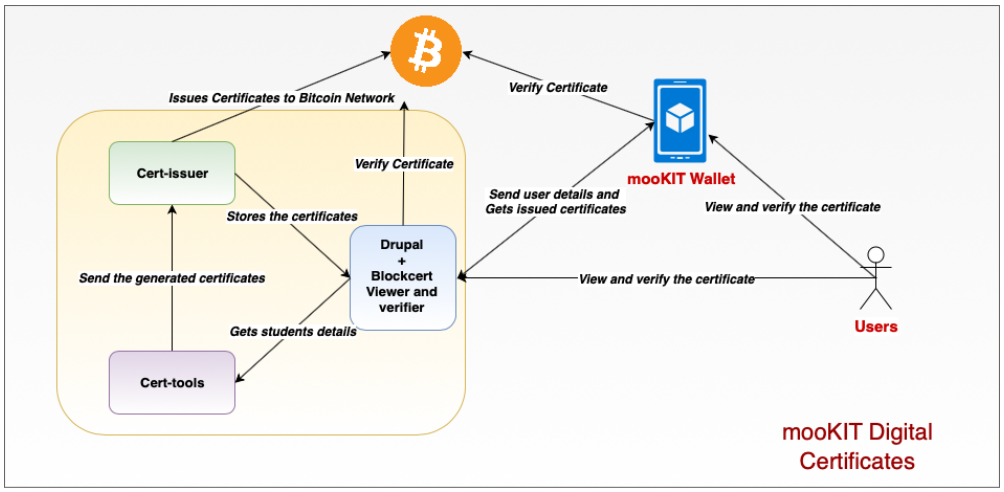

Figure 5. Components of mooKIT digital certificates

During the setup, a 24-word passphrase is generated which acts a seed for generating a unique public address for each user against which certificates are issued. This passphrase is be used to revive the credentials in case of setting up mooKIT Wallet on another device, which makes safely storing the passphrase necessary.

After the setup, users verify their identities by providing the identification used in the course(an email or a phone number) along with the one-time verification code provided in the invitation. The user's public address is then sent to the server against which certificate is issued.

5.2 Generating certificates and making transactions

5.2.1 Generating certificates from templates

Cert-tools[10], an open-source library is used to create templates from which certificate for each student who has verified their identities is generated. The templates have information about the instructors, institutes, course, and the type of certificate being generated.

5.2.2 Making transactions to write certificate information onto the blockchain

Blockchains, being a collection of blocks which are data structures, can hold only a certain type of information. In the case of Bitcoin blockchain, the blocks are a collection of transactions. So the certificate information has to be embedded in the form of hash into the transaction which is then mined by a miner after paying the transaction fee. mooKIT digital certificates use Cert-Issuer[11], an open-source library to make the transactions. Since some transaction fee has to be paid every time a transaction is made, the certificates are issued in batches. Cert-Issuer computes the Merkle root by creating Merkle trees using all the certificates in the batch. The Merkle root is embedded in the transaction using a special type of transaction called DATA OUTPUT. Once the transaction is successfully mined, users are notified about the certificate in the app and via the email. A JSON file which carries all the information is sent to the user.

5.3 Verification

As mentioned earlier, one of the challenges with MOOC certification is, students are dependent on issuers to get the certificates verified. Since the user possesses all the information about the certificate in a JSON file, which can be used to verify with anyone following Blockcerts standards, the issuer dependency is removed. Cert-Verifier is used to verify the certificates. The following section discusses the steps involved in verification.

5.3.1 Verification procedure

a. Validate the Merkle proof in the certificate - The Merkle root for a batch of certificates of which a user's certificate is a part of is computed using the Merkle paths present in the certificate file.

b. Compare the computed hash with embedded local hash - Using the user's public address and the other grade information present in the certificate, a hash is computed and compared with the hash embedded in the certificate.

c. Compare the Merkle root value with the value embedded into the blockchain transaction - The Cert-Issuer extracts the transaction information using Blockchain explorer APIs and compares the Merkle root computed in step *a* with the Merkle root embedded in the transaction.

d. Check for the authenticity of the certificate - The authenticity of the issuer's profile is verified by comparing the date at which the issuer's public address is created against the date at which the transaction has been made.

e. Check if the certificate has been revoked - The certificate is then checked for revocation by comparing it against a list of revoked certificates available at a standard HTTP URL.

If someone tries to tamper certificates, the integrity changes which causes the entire verification process to fail, thus making these blockchain-based certificates tamper-proof.

Since all the information is present with the user and on the blockchain network and not with the issuer, dependency on the issuer to get the certificates verified is removed.

We have successfully issued Blockchain-based digital certificates in several courses. Here is a list of them:
1. Blockchains for managers - Introduction and applications
2. Fundamentals of Agricultural Extension
3. Resource Managements in Rain-fed Drylands
4. Basics of C Programming
5. LifeSkills for Engineers
6. TELMOOC
7. Integrated Pest Management
8. Functional Foods: Concept, Technology and Health Benefits

The community of recipients is wide and varied, from Agriculture Students, Computer Science students to Managers. The response has been enthusiastic, even though many of them did not fully appreciate the implications.

6. CONCLUSIONS AND FUTURE WORK

MOOCs act as a platform to deliver the highest quality educational resources to masses across the world, especially useful in developing nations. Since the communication infrastructure is flaky(low bandwidth, expensive, intermittent) novel delivery methods help in increasing the reach. In this paper, we have discussed some innovations in the delivery of MOOCs and the details about how mooKIT uses blockchains to make certificates tamper-proof and remove issuer dependency.

As MOOCs are centered around videos, technological solutions to stream videos over low bandwidth can be beneficial to the students.

We would like to develop Offline mooKIT as a service for OER, where anybody can build their apps just by filling forms and deliver courses to students.

With the wider adoption of Blockchain technologies, certificate on Blockchains will become more commonplace, and one needs to work on simplifying the process both at the programming level as well as at the user interface level.

REFERENCES

1. mooKIT, http://mookit.co
2. Drupal, Retrieved on June 20, 2019, from https://www.drupal.org
3. Markku Laine, Kalle Säilä (April 20, 2012), Performance evaluation of XMPP on the web, from https://pdfs.semanticscholar.org/23f8/5450ab0cec26bd2e72ccaa09704682d79dcd.pdf
4. XMPP, Retrieved on June 20, 2019, from https://xmpp.org
5. eJabberd, Retrieved on June 21, 2019, from https://www.ejabberd.im
6. Firebase Cloud Messaging, from https://firebase.google.com/docs/cloud-messaging
7. Blockcerts, Retrieved on June 24, 2019, from https://www.blockcerts.org
8. BIP39, Retrieved on June 24, 2019, from https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki
9. Cert-Verifier, Retrieved on June 25, 2019, from https://github.com/blockchain-certificates/cert-verifier
10. Cert-tools, Retrieved on June 25, 2019, from https://github.com/blockchain-certificates/cert-tools
11. Cert-Issuer, Retrieved on June 25, 2019, from https://github.com/blockchain-certificates/cert-issuer