A MOBILE CROWDSOURCING SYSTEM FOR FOOTBALL MATCH LIVE VIDEO STREAMING

Georgi Iliev[1]

Abstract: Mobile crowdsourcing is a fast-growing emerging approach whereby large groups of mobile users are engaged in a collaborative work on performing a particular task or using its results. This paper presents a concept for the development of a mobile crowdsourcing system with extended capabilities for real-time broadcasting and receiving amateur football match video. It is designed to resolve the problem of possible delays and the overload of a system and to accelerate the process of big video data transmission. The proposed system is based on a service-oriented, three-layer cloud architecture and a specialized mobile video streaming application. The architecture includes a main server, infrastructure of scalable multi-parallel video processing engine and an auxiliary server for synchronizing real-time information, which significantly facilitates the handling of user requests with minimal cost and at a high speed. The concept is realized in the Footlikers platform as a basic client-server, WOWZA streaming engine, deployed on an Amazon EC2 cloud machine and a simple sync-server. The results of the program realization of the developed system prototype are presented, regarding football game video steaming intended for amateur football competitions based and organized in France, Belgium and Luxembourg.

UDC Classification: 004.77; **DOI:** http://dx.doi.org/10.12955/cbup.v6.1298

Keywords: crowdsourcing, three-layer architecture, Representational State Transfer (REST), video streaming, video codec.

Introduction

The expansion of ubiquitous mobile communication technologies in all aspects of human activity constantly creates new and growing user requirements for software and hardware developers. Modern mobile devices (smartphones, tablets) provide excellent high-quality capabilities for picture and video capture, GPS, sensing and recording, combined with the power communication technologies that facilitate the transfer of this type of information over the World-Wide Web (WWW). When such a type of activity is carried out by an interested group of people, the term mobile crowdsourcing is applied. In the most general case, the paradigm of crowdsourcing involves the use of potentially large crowds of participants either paid or unpaid to perform diverse tasks of different types - from solving research problem and online marketing to collecting sensor environmental data or the sharing of information. Classic examples of crowdsourcing are Wikipedia, and YouTube for creating collective resources. Other well-known examples are the human computing platform Amazon Mechanical Turk (https://www.mturk.com/) which has more than 50,000 'Human Intelligence Tasks' readily available to freelancers who look for work, Clickworkers (https://www.clickworker.com/) with a million workers who collect different data at the request of a customer and deliver it in real time, and others (see (Sheehan, 2018; Korthaus & Dai, 2015). Over the last few years, this emerging approach has become a prerequisite for new developments in favor of business.

In the field of computer science, there is still no solid theoretical basis and exact taxonomy for the notion of crowdsourcing. There exist over 40 different definitions for it in the literature. In 2006 the American journalist Howe defined crowdsourcing as a "business practice that means literally to outsource an activity to the crowd" (Howe, 2006). More detailed descriptions and updates of the term could be found e.g. in (Brabham, 2013; Prpic, 2017). To varying degrees, crowdsourcing definitions include the following components (Korthaus & Dai, 2015): a clearly defined crowd, task and goal; clear recompense received by the crowd; the process of participative type (with work, money, knowledge or experience) is online assigned; there is an open call of variable extent; the use of the ICT.

For leveraging crowdsourcing tasks, architectures, platforms, infrastructure and services in cloud-based environments are developed and are being examined in many studies, including for mobile crowdsourcing. Related review papers (Doan et al, 2011; Yuen et al, 2011) discuss on basic deployed crowdsourcing systems on the Web, on the components and functions of crowdsourcing systems, as well as on their privacy and security. General types of system architecture and morphology for classification of mobile crowdsourcing applications are presented in (Fuchs-Kittowski & Faust, 2014).

Another field of study is video coding and methods of video streaming in different IT-media. Transmitting big video data takes key part in creation and using of specialized software, infrastructure

[1] Faculty of Mathematics and Informatics, University of Plovdiv Paisii Hilendarski, Bulgaria, iliev86@gmail.com

and platforms, especially for live video streaming applications. Overviews of the current state of development and future challenges associated with the processes of video streaming, main architectures and services adopted over the years for streaming live and pre-recorded videos, including the internet of things paradigm are presented in (Pereira & Pereira, 2016) and their cited literature. More additional aspects, including RESTful web-services and software tools designed to create next generation mobile crowdsourcing applications could be found in (Christensen, 2009).

In this work we present some new aspects in the concept of the paradigm for mobile crowdsourcing. The idea includes creating, distributing and sharing live video streaming of football matches with the help of a mobile application and three-layer crowdsourcing system. Asynchronous input of a large number of users into the system and live streaming video streaming over large time intervals requires the development of specialized software for mobile devices (phones, smartphones, tablets, etc.). In intensive traffic, this can lead to an overload of the system, as well as to the deterioration of video streaming quality. Key elements to address these issues are the speed and quality of mass-creation and the use of videos under the limited resources of mobile devices and their operating systems. The solution to these issues in our approach is through the proposed new extended three-layer architecture of a mobile crowdsourcing system and the use of the newest appropriate programming languages and ICT tools.

Problem formulation

We consider the problem of distributing real-time amateur video films of football matches recorded with a video camera on mobile devices to other mobile users. For the transmission of video data over the WWW it is necessary to create a specialized system with a web server for process management and multiple user queries as well as providing good quality and sufficient speed in real time. For this purpose, a key component of the system is a mobile application (APP) for coding and compressing video data via video streaming. The processes of transmitting video data from different mobile sources at different times to different end-users are relatively long and markedly asynchronous and can be interrupted and renewed for various reasons. This requires additional processing to synchronize the broadcasts with the real time and the current time of each football match. Several key issues arise:

- Selection of a suitable video codec that compresses or decompresses digital video;
- Selection of a video streaming server to handle the mass parallel streaming;
- Synchronizing the real time with the current time of the game;
- Taking into account different speeds and qualities of the users' internet connections.

System functional requirements

Below, the main ideas of the present work are described with the example of the prototype of the author's system Footlikers (footlikers.com). The following functional requirements are set:

- The registered users of the platform Footlikers must be able to login in the mobile application. New user account registrations can only be obtained in the website of Footlikers.
- The mobile application supports two types of user sessions:
    - Broadcaster – can stream picture and sound from a camera and microphone of their mobile device by using the mobile application
    - Viewer – can watch the live video feed on their mobile device inside the application.
- The viewer must be able to see and browse the week's match schedule of his favorite football teams. He could select and initiate the reception of any broadcasted match.
- There is no limit regarding the number of viewers. Every user of the mobile APP has access to a built-in search engine and can watch every broadcasted football match without payment.
- The Broadcaster must be able to choose a desired football match from the match schedule, for which he has permissions to broadcast. These permissions are provided via the administrative club panel from the website. When he is ready with the selection, the live stream is launched. The Broadcaster must also receive picture and sound from their own broadcast on their mobile device. He must be able to fully control the live feed, by start, stop or pause functionalities.
- The match schedule is created on a weekly basis in the website Footlikers.
- The live broadcast is only available to users that are using the mobile application.

Additional requirements are set by the specifics of the game. A football match has well defined rules for its statuses and scenarios. The main time stages of any football match are: Match start, End of first-half, Start of second-half, Match end, Start of first additional time, End of first additional time, Start of second additional time, and End of second additional time. In case the winner is not decided, some final possible set of stages are: Penalty session start, Penalty session end, Match end.

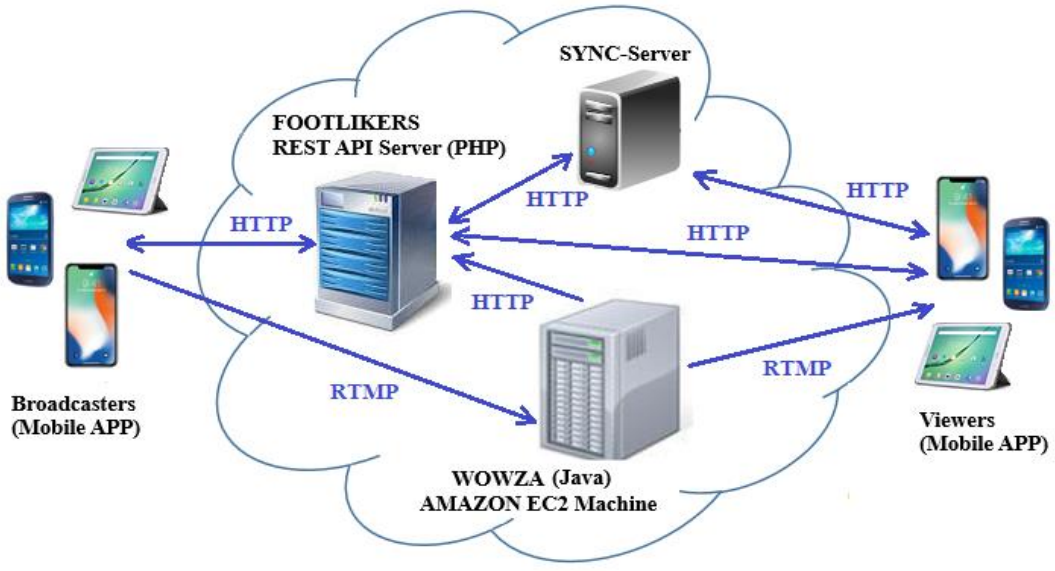
The proposed extended architecture of crowdsourcing system

In general, the basic type of architecture of a crowdsourcing system is linear with the following components: participants for capturing, backend-system (server) for storing and processing the information, and end-users (Fuchs-Kittowski & Faust, 2014). In the case of live video streaming for a large number of users a powerful cloud-computing machine is also required. Such type of infrastructure as a service (IaaS) is provided, for example, by Amazon S3 and Elastic Compute Cloud (EC2, 2018).

At first, we made this more standard and simpler two-layer approach which was typical for any live stream application. The mobile application communicates with the footlikers.com by sending the request using a RESTFUL API service, followed by the initialization and usage of the WOWZA streaming engine providing high scalable capabilities (WOWZA, 2018) to deliver the video stream to all Viewer mobile devices. In fact, any kind of information throughout the football game can be processed by our RESTFUL API. But with such simple architecture we cannot easily synchronize the timing of the match itself along with its many statuses and other aspects of a football game like match interruptions, added time, half-time break, weather factors etc. This includes the actual minute of the game, so that the Viewer can see it on his screen at any time. In order to synchronize the actual minute of the match, all users must send massive number of requests to the REST server. This will greatly overload and overprice the main server because the REST API would be making database queries each time not to mention the general stability of the system could be at risk of crashing.

To prevent all this, the architecture is extended as it is shown in Figure 1, with the inclusion of an additional sync-server. This server itself could be a very small one with only 512MB memory, with the only objective is to handle the current time of the broadcasted game. When the Broadcaster sets the start of the match with status_id =1 (Start match), the mobile device makes a record in the main server via the REST API, followed by an automatic secondary record on the sync-server. This record is not even stored on a database, it is saved on a local text file. For example, <MATCH_ID>.txt to ensure fast read and write manipulations later. Instead of sending requests to the REST API to get the current match status, all viewers will send an additional request to the sync-server by submitting only MATCH_ID.

Figure 1: Three-layer architecture of mobile crowdsourcing system for live video streaming



Source: Author

The additional server then will read the correct text file and will return a simple JSON response with the necessary information to the Viewer. This is by far the simplest possible query there can be, with a size of only 5KB and will return an INTEGER value, which would be the STATUS_ID and the time when the status was submitted by the broadcaster.

After the viewer receives the latest status, then the mobile application will simply adjust the starting point of the match clock which is a simple timer that automatically counts seconds. This method permits the calculation of the clock at any time throughout a football game without occupying almost any memory or hardware resources neither by mobile device nor the server. From this point on, at an interval of 5 seconds the Viewer will send small queries to the additional server to fetch the current state of the game in order to adjust the clock if there are some delays or the beginning or end of a stage. This method also permits the support of different formats of football games and competitions with durations of 15, 20, 30 or 45 minutes even if there are delays of either of the two parties: the match itself, the Broadcaster or the Viewer.

One major disadvantage of the SYNC Server is that its performance is based on the geo-localization of the users. The time to establish connection between the viewer's device and the SYNC server could get longer due to the physical distance between the ISP and the SYNC server or due to DNS Lookup duration which depends on many additional factors: type of connection (WiFi), 3G, 4G etc. An alternative to having a custom made third party SYNC-Server is to distribute the status changes and updates directly inside the RMTP stream connection. This however can overload and destabilize the user's video stream, and it is very limited in terms of resources. From a programmer's point of view, it could be really difficult to upgrade, update, correct and support.

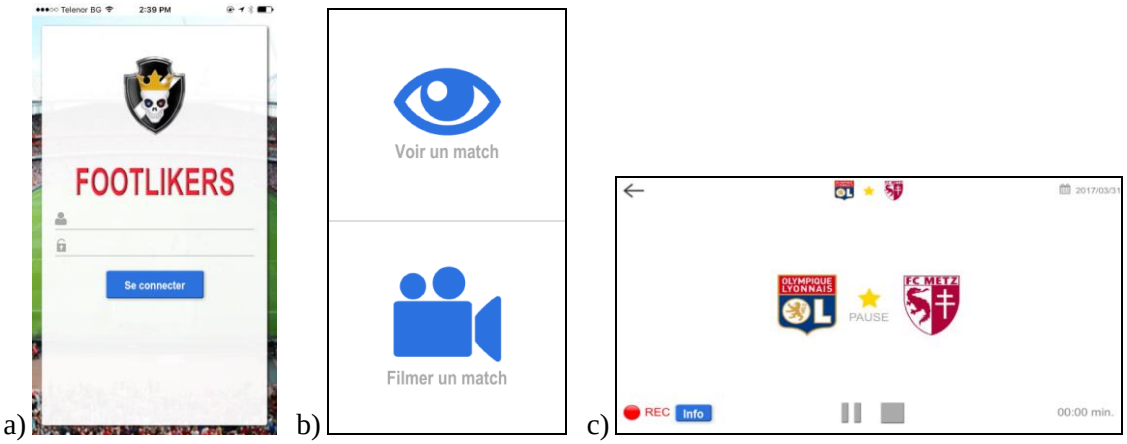
Software implementation

The main software development is based on an ADOBE AIR platform for developing hybrid mobile applications (ADOBE, 2018). The business logic is programmed by ActionScript. It is then pre-compiled and re-compiled to create two versions: one for iOS and one for Android.

When we arrive at the final screen of the application, depending on our session (Broadcaster/Viewer), the mobile application uses NetConnection, which is a native class from the ADOBE AIR Framework, for establishing an open connection to the WOWZA server. Also, NetStream is used via Real-Time Messaging Protocol (RTMP) (Parmar & Thornburgh, 2012) to initialize an audio and video stream connection by means of a Sorenson H.264, a block-oriented motion-compensation-based video compression standard (H.264, 2018). In general, we use the standard build-in framework classes of ADOBE AIR like: APIManager, VideoStream Manager and others.

The login requests towards the RESTFUL API are being handled via a specific URL depending on the type of event and access token which has to pass to guarantee credentials to the API. The RESTFUL API is developed with a LUMEN framework which is the most performant in terms of speed and capabilities (Redmond, 2016).

Figure 2: Screens from the mobile application: a) Login screen; b) Mode selection screen; c) Pause screen, available for Broadcasters only



Source: Author

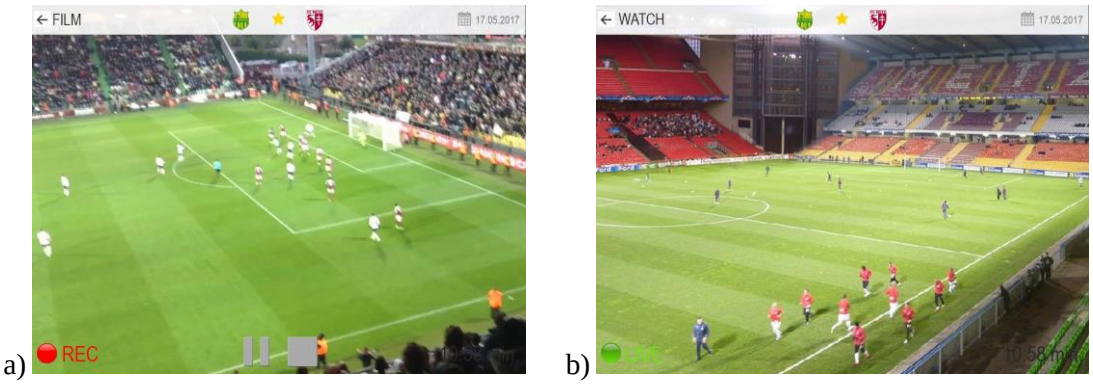
The UI of the mobile application is in French. The login screen is shown in Figure 2a. Then the user chooses his role between the two modes of the application: Broadcaster or Viewer (Figure 2b). When selecting the football match for recording, NetConnection starts and the application will ask for microphone and camera permission from the device, depending on iOS/Android settings of the user.

From this point on we trigger the video stream protocol RTMP and receive/send video and audio data with our mobile device. Figure 3a shows Broadcaster's and Figure 3b shows Viewer's screen views.

As for the built-in search engine of the application it works as follows: we search for "Paris Saint Germain", the engine reacts on each keystroke of the touch screen keyboard and triggers an event which makes a query that locally searches in the application. To make the search engine much more economic in terms of requests, it does not send or receive any data. The whole match schedule of the week is downloaded as a JSON object when the user authenticates. So the search is actually a basic search on a prefetched dataset.

It is possible to choose three different presets of video quality for the broadcast: High, Medium and Low. Each preset has its own different video quality stream setup regarding different video compression and stream parameters. These are stored on the server and can be additionally altered by Footlikers administrators.

Figure 3: Video screens: a) Broadcaster's filming screen; b) Viewer's watch screen



Source: Author

When we click on a preset we send a request to the RESTFUL service and the response is the following JSON code:

```
{
"camFps" : 15,
"camCompression" : 60,
"camMultiplier" : 3,

"codecFps" : 20,
"codecCompression" : 70,
"codecMultiplier" : 0,
"bandwidth" : 0,
"keyFrameInterval" : 30,

"watchBufferTime" : 0.3,
"watchBufferTimeMax" : 1,
"streamBufferTime" : 0.3
}
```

These settings are then applied to the video codec from ADOBE AIR to ensure the appropriate video quality.

Our last example is the Pause screen. It is shown on Figure 2c. It is available to Broadcasters and it is triggered by the pause button on the film UI shown on Figure 2c. Normally there is no "Pause" event on any live streaming applications but in our case, we needed one because our application is intended for amateurs and fans that film with their mobile devices. They often need to move and go to other seats to get a better view of the game. Because we do not want to break the stream since could have many viewers, we propose this functionality. Since in general it is not possible technically, we made it happen with the following principle: RTMP has the possibility to carry and transfer custom service text information. When the Broadcaster clicks the pause, we stop the video/audio data stream of RTMP and populate the connection with internal text data informing the mobile devices that are connection on the stream that the match is on PAUSE. This is just a small text being transmitted on every 1.03 seconds "pause: 1" which triggers an event in the device's UI to show a white screen with

information on it like the team emblems of the two clubs. Therefore, the picture is actually prefetched as well, when the user goes to this screen but by default is in a hidden state. It is then being continuously shown each 1.03 second. This interval is carefully calculated to prevent stream server overload and to ensure that the screen is really frozen. Hence through this way we simulate a Pause without breaking or ending the connection. When the Broadcaster hits the pause button again, RTMP transmits "pause: 0" inside the connection and the video/audio reception is restored.

Conclusion and future work

In the case of a large number of users, providing amateur video, often time-based, with a particular public event, the problem of live broadcasting and receiving of video information of good quality and high speed arises. Such high-tech features can be achieved with the proposed new approach, including the extended crowdsourcing architecture. The architecture can be easily scaled and altered to include many other features as posting comments, sharing moments from the match, transmitting pictures, etc. Possible alternatives for time syncing and underlying infrastructure could be also explored.

One potential future work is extremely interesting. It represents the development of a built-in AI system which scans the video of a recorded video and extracts match statistic from the players that run on the pitch like: number of passes, number of shots on target, distance covered by player, ball possession and many more. With the inclusion of a third additional server, the mobile application can be adapted to handle other sports that are based on strict time frames and intervals like basketball, hockey, etc. Another feature which is being planned is the possibility to stream not only to mobile devices, but also to users logged into the Footlikers website, that will benefit from larger and more convenient screen.

Acknowledgments

This work is partially supported by the MU17-FMI-003 project of the Scientific Fund of the University of Plovdiv Paisii Hilendarski, Bulgaria.

References

Adobe AIR technology. (2018). Retrieved from http://www.adobe.com/products/air.html

Brabham, D. C. (2013). Crowdsourcing. Cambridge: The MIT Press.

Christensen, J. H. (2009). Using RESTful web-services and cloud computing to create next generation mobile applications. In: OOPSLA '09 Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, pp. 627-634. http://dx.doi.org/10.1145/1639950.1639958.

Doan, A., Ramakrishnan, R. & Halevy, A. Y. (2011). Crowdsourcing systems on the World-Wide Web. Communications of the ACM, 54, 86-96.

EC2, Elastic Compute Cloud. (2018). Retrieved from https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/ec2-wg.pdf

Fuchs-Kittowski, F. & Faust, D. (2014). Architecture of mobile crowdsourcing systems. In: Baloian, N., Burstein, F., Ogata, H., Santoro, F., Zurita, G. (eds) Collaboration and Technology (CRIWG 2014). Lecture Notes in Computer Science, vol. 8658, pp. 121-136, Springer, Cham. http://dx.doi.org/10.1007/978-3-319-10166-8_12.

H.264/MPEG-4 AVC standard. (2018). Retrieved from https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC

Howe, J. (2006). The rise of crowdsourcing. Wired Magazine, 14(6), 1-4.

Korthaus, A. & Dai, W. (2015). Opportunities and challenges for mobile crowdsourcing – conceptualisation of a platform architecture. Int. J. High Performance Computing and Networking, 8(1), 16-27.

Parmar, H. & Thornburgh, M. (2012). Adobe's Real Time Messaging Protocol, Adobe. Retrieved from http://www.images.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf

Pereira, R. & Pereira E. G. (2016). Video streaming: Overview and challenges in the internet of things. In: Pervasive Computing, Next Generation Platforms for Intelligent Data Collection, 417-444. http://dx.doi.org/10.1109/FiCloud.2014.18.

Prpic, J. (2017). Next generation crowdsourcing for collective intelligence. arXiv preprint arXiv:1702.03109.

Redmond, P. (2016). Lumen Programming Guide: Writing PHP Microservices, REST and Web Service APIs, Berkely: Apress.

Sheehan K. B. (2018). Crowdsourcing research: Data collection with Amazon's Mechanical Turk. Communication Monographs, 85(1): Special Issue: Advances in Methods and Statistics, 140-156.

WOWZA Streaming Engine. (2018). Retrieved from https://www.wowza.com/products/streaming-engine

Yuen, M.-C., King, I. & Leung, K.-S. (2011). A Survey of crowdsourcing systems: privacy, security, risk and trust. In: IEEE 3rd Int. Conf. on Social Computing, pp. 766-773. http://dx.doi.org/10.1109/PASSAT/SocialCom.2011.203.