Spontaneous Group based Cooperative Live Video Streaming using Smartphones

Juan X. Calles
Univ. Federal do Espírito Santo
Av. Fernando Ferrari, S/N
Vitória – ES, 29060-970
+ 55 (27) 4009-2130
callesjuan@gmail.com

Celso A. Santos
Univ. Federal do Espírito Santo
Av. Fernando Ferrari, S/N
Vitória – ES, 29060-970
+ 55 (27) 4009-2130
saibel@inf.ufes.br

Roberta L. Gomes
Univ. Federal do Espírito Santo
Av. Fernando Ferrari, S/N
Vitória – ES, 29060-970
+ 55 (27) 4009-2130
rgomes@inf.ufes.br

ABSTRACT

Live video streaming through smartphones has proven its potential as a social media, shown in practices of citizen journalism during latest political and social events, such as the Arab Spring, the 15-M movement in Spain, the Occupy movement and 2013 protests in Brazil. A broader coverage of such type of happenings has only been possible by using public live streaming platforms, such as Livestream, Ustream, Bambuser and Twitcasting, each of which provides a mobile video capture application, media streaming servers and a website where viewers retrieve and watch live video content. While video footage is taken and streamed by smartphone devices, such kind of application also meets energy and connectivity constraints. In addition, live coverage of crowded events may be subject to unpredictable or even violent and dangerous situations. On the other hand, public live streaming platforms do not provide enough features for allowing streamers (mobile users capturing and transmitting live video) to interact and cooperate with each other, which could present itself as an approach for coping with unpredictability and smartphone constraints while carrying out a live video stream. In this paper we present an extension for current live streaming platforms where streamers are able to become aware of each other, to form groups and afterwards coordinate and cooperate through additional communication features. A prototype was implemented using Android technology and the XMPP protocol, by piggybacking public live streaming platforms.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *Computer-supported cooperative work.*

General Terms

Design, Experimentation, Human Factors.

Keywords

Live video streaming, cooperative video, social media.

1. INTRODUCTION

Live video streaming through smartphones has proven its potential as a social media, promoting civic engagement on capturing and sharing relevant social and political happenings through live video coverages, carrying out what is also known as citizen journalism. Practices of citizen journalism have been noted within several events around the globe such as the Arab Spring [1], the 15-M movement in Spain [2], the Occupy movements, preceded by the Occupy Wall Street protests [3], and many others. In Brazil we highlight the 2013 protests [4], also known has the June Journeys, were citizen journalism was evidenced through actions of independent and activist journalist networks. One of them is called Mídia NINJA, defining themselves as a decentralized communication network that produces and disseminate content based on collaborative work and online sharing [5].

Current public live video streaming platforms include Twitcasting [6], Livestream [7], Ustream [8] and Bambuser [9], sharing common features in how they enable live streaming. In short, general video streaming architecture consists of the following elements: (1) source/input devices and applications, where video and audio signal is captured, encoded, segmented and streamed to media servers; (2) media servers, where streaming media is stored, retrieved and broadcasted to viewers; and (3) video players embedded inside web browsers, where decoding and playback is taken [10]. Current platforms include, along live streaming services, a website where viewers can navigate and query for live and stored video streams. Also, other common features include a REST API for channel and video querying or even frame retrieval, authentication, geolocation sharing, social networks integration and live social interaction with viewers [11].

Despite its effectiveness as a social media, live streaming through smartphones poses several challenges. While video footage is taken and streamed by smartphone devices, such kind of application meets energy and connectivity constraints. For instance, citizen journalists from Mídia NINJA usually carry USB connected notebooks inside bagpacks for increasing battery life and consequently live coverage duration. Notebooks can also be used for sharing Internet access by plugging a 4G USB, when carrying 3G only compatible smartphones. Another way to increase footage duration consists on asking residents or even store owners, along the path, to share the access to their local wireless networks [12]. Other challenging aspects lie in the fact that live coverages of crowded events are subject to unpredictable and even dangerous situations, like police or protesters violence, which may threaten the lives of streamers (mobile users capturing and transmitting live video). On the opposite side, live streams from large perimeter events can often come across tedious situations, without any relevant developments [11].

Having a broader or global view about what happens dozens or hundreds of meters away, being aware of other live streams taking place within the same event/context and further being capable of interacting with them could be an approach for the aforementioned challenges [13]. Nevertheless, current live streaming platforms do not provide enough features that allow streamers to interact and cooperate with each other. Although geographic location is obtained from mobile devices and some live streaming platforms exploit it, specially Bambuser, which pinpoints their location on a map, it is done exclusively for viewers, not benefiting streamers from provided contextual data. Also, despite enabling a communication channel between each streamer and its own viewers, where feedback or casual interaction takes place, there is no official feature that allows streamers to communicate with each other.

In this paper we present an extension for current live video streaming services which main purpose is to allow streamers to cooperate between them while carrying out a live coverage. Our extension intends to further exploit contextual data to live streamers, allowing them to visualize their own location and points of reference on a map. Next, by sharing their context, streamers should be capable of identifying other streamers and start forming groups in a spontaneous and dynamic manner. Finally, communication features provided to group members should allow them to coordinate and cooperate as means for coping with aforementioned challenges. Initial main communication features are group text chatting and geographically annotated notifications, called *map notifications*. A prototype was implemented using Android technology, the XMPP protocol [27] and reusing public live streaming platforms.

2. RELATED WORK

Of current live streaming platforms, Bambuser, and recently Periscope [14], notably exploit contextual data, both pinpointing smartphone devices' geolocation on a map available to viewers. In Bambuser, the map includes time filters, allowing to visualize streams that are currently live, those that were live 24 hours ago, 2 days ago and so on. It also uses location data to recommend nearby live or stored video, although limited to retrieving streams from the same country. Despite these approaches exploit location data using maps, they do not extended to mobile capture applications, which would allow streamers to become aware of their location, nearby points of reference (streets, parks and buildings) and identify other streamers in the same area. The IStream project [15] also proposes location sharing while live streaming, where users are military and policeman (for security purposes). In this case, video footage and GPS coordinates are not for public access, but instead they are sent to command centers, aiming at increasing their situational awareness about streamers. However, it is not specified how further interaction between the command center and IStream instances is handled, nor if streamers interact directly with each other.

Another feature current public live streaming platforms share is the capability for streamers to interact directly with their viewers, specially through text messaging, which is usually replicated to social networks. Ustream allows a streamer to ask simple "yes or no" questions to his viewers. However, there is no official feature that enables streamers to communicate to each other. Twitcasting actually allows a streamer to navigate through other coverages and even leave comments while carrying out its own stream, however, it is done by reusing the viewers' web interface. This implies that a streamer cannot tell the difference between viewers and other streamers, since he identifies everyone as a viewer. It also makes impossible a many-to-many conversation to be established between three or more streamers.

[16] describes Caleido, a system that actually allows streamers to become aware of each other, using geographical and relative location data that is uploaded along with media content. Caleido uses a simple client-server architecture where the server stores sent data and automatically clusters nearby streamers. The mobile application presents, in its main screen, a typical camera preview and a sidebar showing the number of nearby capturing devices. A secondary screen shows video from a nearby participant and a button that allows navigating through footage from all the other participants. The navigation button is labeled with the relative location of the next video source, such as "100 meters to the right". Viewing footage from other streamers in real time provides a broader view of how an event is being covered. This is useful, for instance, for reducing content redundancy, as one person can detect that higher quality video is being captured by others from the same location/angle. Another offered feature allows to mark content from others as "liked" which, besides providing feedback to each participant, the server uses later to automatically generate a mix, based on most popular videos. Therefore, Caleido promotes cooperative live streaming by increasing situational awareness, although limited to textually describing the relative location of participants, which results in indirect cooperation, as participants cannot actually communicate and interact directly.

[17] proposes sharing video footage for accomplishing cooperative tasks, particularly geocaching, an outdoor activity similar to hide and seek using GPS. Cooperation happens between two individuals, each of them equipped with a custom prototype made from commercial hardware and software, consisting of a camera attached to a pair of sunglasses and a smartphone attached to an armband, both connected through Bluetooth. The camera attached to the sunglasses aims to capture the point of view of an individual, which is transmitted to the smartphone through Bluetooth and finally sent, through the Internet, to the smartphone worn by his partner on the other edge. This way, they can visualize each other's point of view, by checking the smartphone attached to their arms. Thus, this approach uses video as a mean for cooperatively accomplishing a task rather than a goal, although sharing video previews does enable cooperative video production, as seen in Caleido. Audio is also captured and streamed, so users can talk to one another throughout their experience (direct communication). Geocaching pairs are predefined, therefore no pair matching or formation is addressed.

Multiplayer online games (MMOs) can also contribute for cooperative live video streaming, mainly those that present competitive scenarios. Both MMOs and cooperative live streaming have to cope with remote protagonists, or players, that are logically grouped and which together need to achieve a group objective. For multiplayer games, the usual goal is to beat other groups. For cooperative live streaming, the goal can be interpreted as to offer a broader and longstanding video coverage of an event. [18] studies most common cooperation features present in multiplayer games, including real time text and voice communication, and proposes a framework for automatically triggering an interaction according to a player's context. Specific genres, like FPS (First Person Shooter) and MOBA (Multiplayer Online Battle Arena), provide a map or a radar where it is possible to locate team members and enemies. Particularly, MOBA games allow interactions using the map [19], like notifying allies about enemies missing or next targets. These features are the basis for cooperation in our cooperative live streaming extension.

Group formation is addressed in [20] through Flocks, dynamic user groups that are formed according to high level descriptions, such as "is college student" and "is nearby", that can be combined, for example, to retrieve nearby college students. Participants must firstly provide their profile information and then they can define multiple sets of high-level descriptions, each corresponding to a Flock. Matching is done by comparing a Flock's description against profile information from identified users, through Wi-Fi direct. Flocks are managed by the Urbiflock framework [20]. The Group Context Framework (GCF) [21] also allows mobile devices to form groups, based on subscriptions to context data feeds from other GCF enabled devices. The GCF communicates and identifies nearby GCF devices through Bluetooth. Flocks and GCF are adhoc and do not represent groups globally, therefore each participant sees and manages his own groups. Although they provide an approach for grouping streamers, a global and centralized representation would simplify group management, visualization and enable server-side routines, such as monitoring resources.

FOCUS [22] is a system aimed at grouping live video streams based on geographical and relative location data combined with computer vision techniques. Sensor data sent along video content includes geographic coordinates from GPS, compass, accelerometer and gyroscope. Exploiting the relative camera location and orientation of two or more streamers, FOCUS traces their lines of sight and identifies geometric intersections, being able to infer content similarity even when video contains little or no visual similarity. FOCUS uses a technique from computer vision called "structure from motion" and, for each stream, it develops an estimated 3D model of the user's line of sight. It then compares pairs of streams, frame by frame, looking for line of sight similarity. Finally, comparing multiple streams and multiple frames, a spatio-temporal matrix of content similarity is built and, applying clustering techniques, it returns groups of streams with shared subjects. Thus, FOCUS goes beyond grouping streams in the same event, identifying those who captured the same objects. It does not address, however, cooperation between streamers or even increasing situational awareness once groups are identified.

Other attempts to group live video come from websites that explicitly aggregate feeds from live streaming platforms like Ustream and Livestream. One of them is OccupyStreams [23], which aggregates channels dedicated to streaming social and political content, as its name is in reference to the Occupy movements [3]. Channels are added manually and grouped by tags according to their subject. Another similar website is Web Realidade [24], dedicated to Brazilian citizen journalism. Channels are also added manually but it additionally organizes them according to their location. Live streams are displayed in a grid based on their country state and, when clicked, a bigger video player pops up for the selected stream.

Finally, other contributions to cooperative live video deal with live editing and mixing. IBS [25] is a system that supports up to five live video feeds from mobile capturing applications. The user (director) can simultaneously view all five mobile camera feeds and choose one to go on air. When a particular feed is selected, its owner is notified through a back channel. Additionally, the back channel allows the director to send text messages to streamers, allowing live coordination, although centralized and only in one direction. One interesting fact mentioned is that text messages tend to distract broadcasting streamers, but it is useful for planning and coordination before capturing video or during intervals. It also states that increasing situational awareness can reduce content redundancy, as an editor can inform a streamer about other nearby streamers. Another tool is the MVM [26], a mobile-based mixing application, fetching up to four feeds from Bambuser. It also piggybacks Bambuser's capture application and streaming servers. No cooperation features are provided by MVM.

3. COOPERATIVE LIVE STREAMING EXTENSION

Here we present our extension for cooperative live video streaming through smartphones. Our main goal is to allow streamers within the same event to actively cooperate while carrying out live coverages by providing them with additional features that allow them to communicate and coordinate. We expect that through cooperation some issues regarding resource constraints and unpredictability from live coverages can be addressed. To achieve cooperation the first step is to provide participants with awareness of their surroundings, mainly including other streamers, through what we call *context provision and awareness*. For this, later we present a concept called *context sharing*, which represents a session where contextual data is publicly shared by a streamer. Next, using context data, specially geographic location, the second step is to allow streamers to dynamically and spontaneously form groups. We propose that groups are not formed automatically but actively by streamers. In fact, each streamer can choose individually to add these new features or maintain a typical live stream. Once a group is formed, cooperation features are made available for its members into the third and final step.

3.1 Architecture

In short, video streaming consists of capture, encoding, streaming and playback [10]. For live streaming through smartphones, platforms such as Ustream and Livestream provide a mobile application where video is captured, encoded and streamed to streaming servers, also provided by these platforms on their server side. Furthermore, they also provide additional services such as authentication, social interaction and a website where content can be queried and browsed. When viewers, navigating the website from a web browser, select a particular live feed, the streaming server then starts streaming video to the viewers' video player attached to the browser, where finally playback takes place [11].

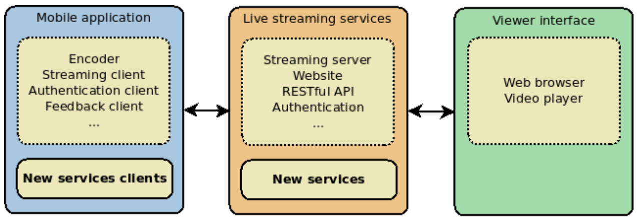


Figure 1: Overview architecture

As an extension for current live streaming platforms, we propose adding new features through an additional set of services, without necessarily interfering with current platforms' architectures. Figure 1 shows an overview of common live streaming services already offered and, bellow them in bold text, the new set of services for cooperative live streaming (and their clients). New services are independent of existing ones (specially from streaming servers), although eventual integration between them would be convenient, as for authentication. We present later, in the prototype section, that independent services allows us to piggyback aforementioned live streaming platforms, implementing context awareness, grouping and cooperation concepts on top of them. Data from these additional services will be exchanged through a data channel parallel to media itself, similar to how social interaction and feedback data is currently handled.

Concretely, we propose adding the services shown in Figure 2, along with their clients. First, a service named *context and group mapper* (CGM) will be responsible for handling context provision and awareness, which will be later used by streamers to form groups. Next, we propose a set of services called *cooperation services* that will actually provide cooperation features for streamers within a group. Initially we propose two such types of services, a *group chat service*, for direct textual communication between members of a group, and the *map notification service*, for sending geographically tagged messages, called *map notifications*, which carry specific semantic meaning, such as "let's meet at this location". Cooperation services are based on video game cooperation features [18][19].

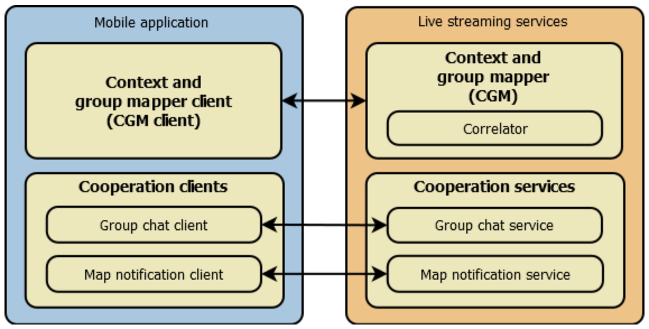


Figure 2: Architecture

3.2 Context provision and awareness

Context provision is represented through what we call a *context sharing*, a type of session where context data is uploaded from mobile devices, through the CGM client, to the CGM on the server side. Context shares are finite, having a starting and an ending time. During its life cycle a context sharing can be paused if desired or it is automatically paused when an error occurs, like connection loss, and can be later continued. When a context sharing is finished by the streamer, the only way to keep providing context data is by starting a new one. We have also defined context shares as being unbound from video streams, so it is possible to interrupt a context sharing while maintaining a video stream or vice versa, depending on specific requirements, such as low battery. However, it would be ideal to have both running simultaneously.

Context provision is handled by the CGM (which also handles grouping) and the CGM client, on the server side and mobile application respectively. The CGM serves as a repository that stores context and group data that is mapped to a live stream. Context data is gather by the CGM client and sent to the CGM while there is an active context sharing. Initial context data we represent are: geographic coordinates, relative camera orientation, hashtags, timestamps, battery level, device status (like low battery or bad Internet connection), streamer status (like injured or detained), etc.. Ratings from viewers also represent information about a streamer, like popularity, camera skills, camera quality, etc. We also anticipate the use of ratings from other streamers, which mainly express a streamer's cooperation level. Context data can be automatically monitored by the CGM or by its client. For instance, the CGM client can monitor battery levels and, when reaching a certain threshold, it can request the CGM to update the device status to "low battery" and even send a *help notification* (a type of *map notification* that will be presented later). Furthermore, the CGM can monitor cooperation levels by, for example, verifying responses and streamers' movement that follows a *help notification*.

Context data is exchanged by messages sent from the CGM client, on the mobile application, to the CGM. Two types of messages are defined: *life cycle messages* and *context update messages*. *Life cycle messages* are associated to status modifications on context shares. An *initiate message* is called actively by a streamer for creating a new context sharing, setting its status to *active* and starting sending periodic *context update messages*. A *pause message* can be called manually by the streamer, or it is called automatically when errors occur, such as lost connectivity, changing its status to *paused*. *Context update messages* are temporarily interrupted when a context sharing is paused. A context sharing becomes active once again when a *resume message* is called actively by the streamer, changing its status back to active and sending periodic update messages once again. The *terminate message* permanently terminates a context sharing together with the context update messages that it sends. It is called manually, but it can also be configured to automatically terminate a context sharing when paused for too long. The *status message* retrieves the current status of a context sharing, so the CGM client can prevent sending any wrong messages, like sending *context update messages* while paused or resuming a context sharing that was automatically terminated. *Context update messages* are only sent when a context sharing is active and they are sent periodically. Context data that can be provided manually by the user, like hashtags, or automatically through implemented sensor listeners, especially for location and position sensors like GPS, accelerometer, magnetometer and gyroscope, which together provide geographic coordinates and relative camera orientation. Current implemented context update messages in our prototype are *update hashtags* and *update location*.

Context awareness is mainly exploited by providing streamers with a map that plots their own geographic data, like geographic coordinates and relative orientation, as well as data from group members. Providing this information to streamers increases awareness about their current location and its surroundings, specially when they have no previous knowledge of it. They can, for instance, visualize points of reference near them, such as parks, streets and buildings. Also, displaying the location of group members allows each streamer, together with cooperation features described ahead, to plan and coordinate actions for live streaming, such as avoiding content redundancy or setting targets, etc.

3.3 Spontaneous and dynamic groups

The next step is grouping streamers within the same event. Groups are formed based on context shares and data that is available on the CGM, but they are not formed automatically. Instead, groups are formed actively by streamers when desired in a dynamic and spontaneous manner. This is mainly an approach for avoiding constant rearrangement caused by streamers' variable availability, for example, due to smartphone connection losses or battery depletion. It also allows streamers to distinguish between events in the same geographic location, overcoming the use of simple distance boundaries. It can also simply allow streamers to decide freely who they want to team with, for example, according to their political orientation. We do not address access restrictions or group moderation in this paper.

When a group emerges it contains only one member, as every new context sharing creates a new group, with its status set to active. A group grows when other members start joining it, and diminishes when they leave. When a group is left with no members its status changes to inactive permanently. The incoming and outgoing of members of a group takes place actively through messages sent to the CGM, these are the *find groups*, *join group* and *leave group* messages. For a streamer to join any surrounding group, he initially needs to become aware of them. This is done by requesting the CGM to retrieve active groups within the same context, particularly those in a nearby location, which is accomplished through the *find groups message*, retrieving, for each group, its group id, aggregate hashtags and current number of members. Next, a streamer can join a group by calling the *join group message*, which requires its group id. When a member joins a group, he automatically leaves his previous group. Also, right after joining a group, streamers receive information about its current members, which can also be requested actively later, by calling the *fetch members message*. A streamer can also leave a group when desired, by calling the *group leave message*, leaving his current group and creating a new one. For each group the CGM can perform simple operations on data from its members, like aggregation or summarization, for example, aggregating their hashtags and periodically calculating a group's centroid based on the geographic coordinates from all its members. The CGM can also monitor the cooperation between members of a group.

The CGM has a special element called the *group correlat*or, which is a piece of software that periodically scans data inside the CGM looking for clusters of groups near each other, using their geographic coordinates represented by their centroids. Additional filters can be added to further refine found clusters, as a mean to avoid undesired correlations. For instance, after finding clusters based on geographic proximity, the correlator can verify if groups inside a cluster also present related hashtags. As a result, a cluster could be split into two or more smaller clusters. After a cluster is identified, the correlator selects one of its members, a group, to be the pivot group. A pivot group will be taken as the reference group and it will be suggested to all the streamers from other groups within the same cluster. Suggestions are sent through a message called *group suggestion*. When a streamer receives a group suggestion, then he can actively choose whether to join it or simply ignore it.

3.4 Cooperative live streaming

In order to have cooperative live streaming through smartphones, we provide groups with a set of features that allow streamers, within the same group, to communicate and coordinate actions in real time while carrying out live coverages. These features are achieved by *cooperation services* and *cooperation clients*, shown back on Figure 2. A cooperation service comprises the software and infrastructure needed to exchange messages between a group, for a given cooperation feature, delivering a message from a streamer to all the other members of its group. Current cooperation services are *group text chat* and *map notifications.* The cooperation client, in the mobile live streaming application, allows a streamer to compose, send, receive and visualize messages, using a specific cooperation service. These cooperation services are mostly based on cooperation features present on multiplayer online games, most of them including a map or radar visualization and interaction along text and voice chatting [18], each with its advantages and disadvantages for accomplishing a specific purpose. What they all have in common is the need for communication to be carried in real time, hence, real time communication protocols become useful when providing cooperation services. Specifically, in our prototype we implement cooperation services based on the XMPP protocol, where the server-side is basically a Jabber/XMPP server and cooperation clients are XMPP clients.

The first cooperation service is *group text chatting*, which enables members of a group to communicate and coordinate in real time by exchanging text messages. Text messaging while live streaming is already available for most current live streaming platforms, although it is only available between a streamer and his viewers. The group chat service intends to provide streamers, members of the same group, with a communication channel between them. The main advantages of text messages are their expressiveness, small sizes and simplicity. As a disadvantage, texting while streaming might be slow and distracting, as a streamer might be focused on capturing relevant footage, resulting in shaky video or even constant floor footage, as it is common to position a smartphone's camera facing to the floor when composing text messages. On the other side, during protests, streamers like Mídia NINJA already interact with their viewers through text, proving certain familiarity with this type of feature and opening a breach for a streamers' group chat.

Map notifications, the second cooperation service, are less generic and provides less expressiveness, but they embed geographic information and carry a well defined semantic meaning. Basic usage consists of a streamer willing to notify a very specific information using a map. Firstly, he selects the type of notification to be sent, then he provides its geographic coordinates, by selecting a point on the map visualization present in the mobile application, and lastly provides a short text description. The notification is then sent to all group members and, when received by the mobile application, it is displayed on their map visualization, using provided geographic coordinates. Map notifications are temporary, disappearing after a while.

Four initial map notification messages were designed: *target notification*, *danger notification*, *help notification* and *moving notification*, all inspired in online multiplayer games [19]. The *target notification* is meant for streamers to notify others about relevant happenings within an event (at provided geographic coordinates) that might be of interest for the group, for example public pronouncements or denouncements and even meeting locations. Figures 3 helps in representing its semantic meaning, where each colored circle represents a streamer and the attached yellow triangles represent their fields of view. Streamers that send the map notification are colored in green. Figure 3.a presents the moment when a target notification is sent, located at the green mark and text description is something like "meet here". Figure 3.b shows how other streamers are expected to behave according to its semantics, which is moving towards that location.

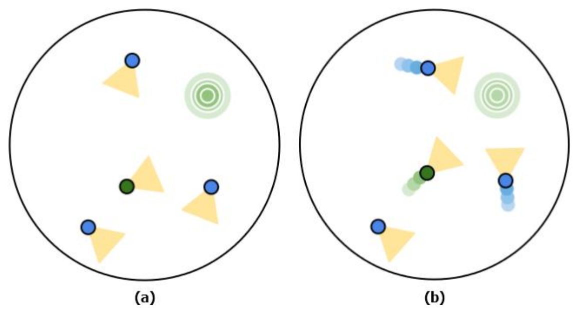


Figure 3: (a) green streamer sends a target notification; (b) streamers move towards the notified location

The *danger notification* notifies a dangerous situation during an event, so members of a group can avoid them (specially when their lives are in threat). Figure 4.a shows a streamer that notifies a dangerous situation on a given location (red mark), like police or protester violence, providing a text description “confrontation”. Figure 4.b shows that, once other streamers become aware of danger, they retreat and keep a safe distance from that particular location, including the streamer that sent the notification.

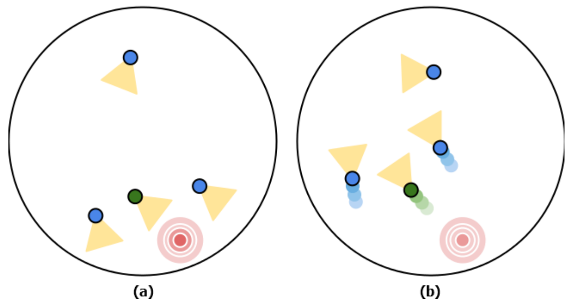


Figure 4: (a) green streamer sends a danger notification; (b) streamers close to the danger turn away

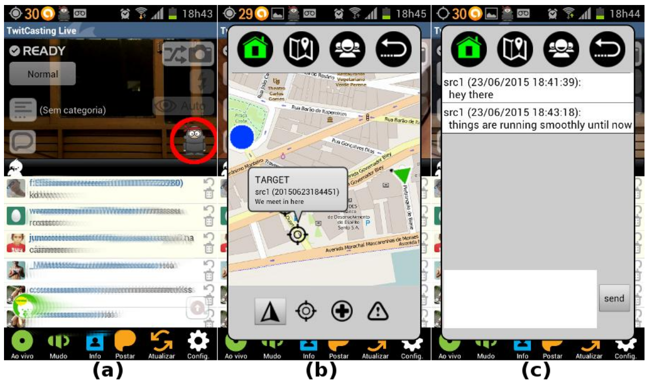
The *help notification* is used to request help to nearby streamers and its geographic coordinates are the same as the requester’s. Requests for help range from technical problems (low battery or requests for Internet sharing) to physical threats (getting injured, cornered or arrested), which is specified in their text description. Help notifications can also be automatically sent by the CGM client when, if monitoring battery levels, it reaches a certain threshold. Map notifications, and specially help notifications, can initiate a cycle of cooperation monitoring. For instance, when a help notification is sent, the CGM can start monitoring streamers’ movement, group chat and map notifications that follow, verifying if they interact or move towards the requester, and then automatically rate their cooperative behavior.

The *moving notification* is for notifying a streamer’s future location. This has two main purposes, the first one refers to coordinating a cooperative live coverage, as for when a streamer is aware of the current and future location of his partners, he acquires more information for deciding his next actions, like avoiding locations already covered. The second purpose is more specific, as a moving notification can be used as a response to a help notification. When a streamer receives a help notification and he decides he will respond to it, he sends a moving notification using the geographic coordinates from the requester. This way, the requester receives a feedback and acknowledges that a person will attend him, also other members of the group become aware that that particular request was attended.

4. PROTOTYPE

Having context shares and video streaming unbound from each other allowed us to piggyback live streaming platforms and focus exclusively in implementing services from our cooperation extension, particularly the CGM and the group chat and map notification services. On the server side, the CGM was implemented as a daemon based in Python, handling *context and group messages* and running the *group correlator*. Received context and group data is stored in a MongoDB database. The Openfire XMPP server was used for cooperation services, as its Multi-user extension is all that is needed for allowing the group text chat and map notifications, associating each streamer to a Jabber ID and using Multi-user chat rooms to represent groups. As it was early decided to use the XMPP protocol, access to the CGM is also XMPP based, specifically through IQ messages, using the SleekXMPP library. IQ messages behave much like common HTTP requests/responses, which could have also been used for accessing the CGM.

On the client side, we implemented an Android application providing the cooperation features for streamers. Clients were implemented using Android foreground services and the Window service for the UI, instead of a regular Activity application. This allowed the prototype to run on top of any existing live streaming application, including Twitcasting (widely adopted in live coverages of protests in Brazil). Figure 5.a shows the prototype overlay icon (circled in red) running on top of Twitcasting and, when touched, an overlay window pops-up, containing actual cooperation features (Figures 5.b and 7.c).



TARGET
src1 (20150623184451)
We meet in here

src1 (23/06/2015 18:41:39):
hey there

src1 (23/06/2015 18:43:18):
things are running smoothly until now

Figure 5: (a) overlay icon; (b) map view; (c) group text chat

The application allows a streamer to manage a context sharing’s lifecycle and to find and move between groups. The map visualization (Figure 5.b) was implemented using the OSMDroid library, which uses the OpenStreetMap geographical database. The map displays streets and points of reference, but specially one’s current location (green triangle), group members’ location (blue circle) and map notifications (target icon). A map notification is added by tapping its type (icons in the bottom), and then its location in the map. Map notifications and group chat XMPP clients were implemented using the Smack [28] library. A simple chat view (Figure 5.c) is used for the group chat, featuring a list stacking exchanged messages ordered by the last received, a text input for composing a new message and a send button.

5. EXPERIMENTS

Three experiments were conducted in order to: (i) functionally test the implemented prototype; (ii) *assess* its impacts on smartphones' resources, specially on battery use and transferred data; and (iii) measure the average communication delay, using common Internet access services in town (3G and Wi-Fi). The first experiment tested functionality. It was performed by three volunteers in Vitória city's downtown, within a 120 meters radius distance. Devices used by volunteers were a Samsung I9100 and two HTC One M7, all using 3G connections at 1Mbps. The experiment consisted of executing a simple predefined scenario, if the scenario was concluded then it was considered success, else it was considered failure.

The scenario consisted of the following: (1) volunteers randomly chose a starting location and one among them to be a *notifier* (the person that would trigger a map notification), without others knowing; (2) volunteers walked towards their designed starting locations; (3) upon arriving, each one immediately had to first start a new context sharing, using our prototype, and next a new video stream, using any video capturing application (they were asked to capture anything they wanted); (4) next, they were instructed to accept the group suggestion sent by the group correlator, as we expected to test if a group containing all volunteers would be successfully created; (5) on joining the suggested group, they had to immediately use the group chat to greet others, until they became aware that all the volunteers had already joined the group; (6) finally, the previous randomly selected *notifier* would send a target notification providing his current location, where all the participants would meet and the experiment terminated. The experiment succeeded if all these steps were achieved. This experiment was performed four times and, for each one, starting locations and the *notifier* were selected randomly, through the first step.

The second experiment was the resources' usage. It consisted of measuring the time it took to drain five percent of battery level, using six different running configurations, three times for each one. For each test we had the device used for the experiment charged at a hundred percent battery level and each one was finished when the battery reached ninety-five percent value. The amount of bytes sent and received was also measured. For this experiment we used a Samsung I9100 with an Android Ice Cream Sandwich operating system. Also, simulated streamers were implemented in Python to easily simulate group interaction and test its impact in battery consumption. Simulated streamers were able to send and receive *context and group messages* and even *group chat* and *map notification messages*.

The first configuration, or default configuration, was to only run the streaming application, without using our prototype, to estimate reference values of battery usage without additional cooperation features. We used Twitcasting due to its popularity in coverages from Brazil's protests. The five remaining configurations used context shares and, for each one, we also stored the amount of exchange data (sent and received) with the CGM and other simulated streamers. The second configuration was to simply run Twitcasting and an active context sharing simultaneously.

The third configuration added five simulated streamers, forming a group of six members. In this case, no cooperation messages were exchanged, except for Presence messages from the XMPP protocol. The fourth configuration also included five simulated streamers, but this time, they were configured to send a group chat message every minute, simulating interactivity. The fifth and sixth configurations were similar to the third and fourth respectively, but, instead of five, ten simulated streamers were used.

The third experiment tested communication delay. Particularly, the *group suggestion*, *join group* and *map notification* messages were tested, mainly because of their important role in group formation and cooperation, as seen in the first experiment. For this experiment, we used a Samsung I9100 and a simulated streamer. Delay was tested for both Wi-Fi (10Mbps) and 3G (1Mbps) networks. Additionally, 3G networks were tested in two different environments, according to signal strength (weak and good). The experiment would first start the simulated streamer and then the actual streamer. The actual streamer would receive a *group suggestion* and then join the simulated streamer's group. Finally, the streamer would send a *map notification*. During this process, which was repeated ten times (for each access service), communication delay values were stored for each of the previously mentioned messages. We used the round-trip delay time to measure delay, mainly for its simplicity.

5.1 RESULTS

For the first experiment, all four attempts were successful on completing the predefined scenario. As a general feedback, users stated they did not experience any difficulties in handling the prototype. In fact, during the second attempt, the correlator was not initiated correctly by mistake and thus group suggestion messages were not sent. However, as participants were first introduced to the prototype features, when they realized that the *group suggestion* had delayed, they were able to manually use the *find groups* and *join group* features, successfully forming a group that contained all volunteers.

On the second experiment, Table 1 presents the average time spent and transferred data for each configuration. Considering the sixth configuration to be the most extreme among the others, the greatest loss of coverage duration when using our prototype corresponded to approximately 13.4% of the default configuration. The second worst time loss happened on the fourth configuration, corresponding to 11% of the default. Both the fourth and sixth configuration were also the ones to exchange the greatest amount of data, 40.4KB and 50.8KB respectively.

Table 1: Battery and data experiment values

Configuration	**Default**	**2nd**	**3rd**	**4th**	**5th**	**6th**
Time	9min16s	8min41s	8min33s	8min14s	8min26s	8min1s
Sent (KB)	-	4.5	4.1	4.9	4.2	4.0
Received (KB)	-	4.2	7.7	40.4	9.0	50.8

Although the other configurations did not dealt with continuous data transfer, they still spent battery faster than the default configuration. Some known implementation details that influenced on higher energy consumption were the GPS, automatic XMPP Presence messages and frequently using the vibrator of a device when receiving group messages.

Table 2: Communication delay values

Message	**Group suggestion**	**Join group**	**Map notification**
Wi-Fi	177ms	54ms	46ms
3G (Good) delay	268ms	171ms	229ms
3G (Weak) delay	1021ms	1231ms	959ms

Table 2 shows the average delay for each message on the third experiment. In Wi-Fi and 3G networks, with a good signal strength, communication was considered as almost instant, where Wi-Fi performed better than 3G. In the 3G network, with a weak signal strength, average communication delayed was approximately one seconds. Although not shown in the table, weak signal strength also led to a higher standard deviation, as some messages reached more than four seconds delay. Also, during the experiment, connection with the CGM was lost once.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented an extension for current live video streaming, providing it with cooperation features. The extension is based on groups that are formed dynamically and spontaneously according to context data provided along each live stream, specially their location. A map visualization is presented to streamers, showing the location of group members, but also their own location and nearby points of reference. Cooperation features are available for members of a group, specifically a group chat and map notifications, which allows them to communicate and coordinate while carrying out live coverages.

A prototype was implemented for Android devices, using the XMPP protocol as the basis for communication between group members. Three experiments were carried out, stating that: (i) the prototype works and the proposed features allow streamers to cooperate; (ii) when used intensively, the prototype can increase battery consumption up to 13%; and (iii) in appropriate environments, communication delay usually does not surpasses one second. Therefore, additional features proposed in the live streaming extension are capable of enabling cooperation between streamers, with fairly low impact on device resources. Network coverage still greatly influences on performance, although, if signal is strong enough for members to communicate, they can coordinate a way out, like offering routing services or notifying locations with better signal. In general, streamer motivation and participation still needs to be addressed.

In future work, we intend to approach media activists and receive a detailed feedback on the use of our prototype at crowded events, also about concrete needs of live video streaming that were not addressed by our extension. We intend to further study user interface design aspects that may simultaneously promote cooperation and smoothen streamer's interaction while handling a live coverage. We will also focus on how to present a cooperative live coverage to viewers, exploiting map visualizations, grid views and multi-camera live mixers. Finally, we will study adding a voice chat, which might be more intuitive but also challenging when dealing with group conversations.

7. REFERENCES

[1] Bengtsson. R. Action! Livestreaming as means of civic engagement: A case study of citizen journalism in Egypt and Syria. In *Glocal Times 19* (2013).

[2] Camuñas. M. Miradas colectivas: veinte años de videoactivismo en España. In *Videoactivismo. Acción política, cámara en mano* (2014). 90-94.

[3] Costanza-Chock, S. Mic Check! Media Cultures and the Occupy Movement. In *Social Movement Studies*, vol. 11 (2012), 375-385.

[4] T. Almeida, A. Evangelista. Tecnologias móveis, mídias independentes e coberturas de mobilizações sociais urbanas: as influências do "midialivrismo" na sociedade midiatizada. In *Anais do II Colóquio Semiótica das Mídias*, vol. 2 (2013).

[5] Oximity. Mídia NINJA. https://ninja.oximity.com/ (07.04.2015).

[6] Twitcasting. http://twitcasting.tv/ (01.06.2014)

[7] Livestream. http://original.livestream.com/guide/livetv (01.06.2014)

[8] Ustream. http://www.ustream.tv/explore/all (01.06.2014)

[9] Bambuser. http://bambuser.com/broadcasts (01.06.2014)

[10] Austerberry, D. Introduction to streaming media. In *The Technology of Video and Audio Streaming* (2005). 131-147.

[11] Juhlin, O., Engström, A., Reponen, E. Mobile broadcasting: the whats and hows of live video as a social medium. In *MobileHCI '10* (2012). 35-44.

[12] Hessel, C. No meio do redemunho. *O Estado de S. Paulo* (2013). http://www.estadao.com.br/noticias/1050880 (07.04.2015)

[13] Juhlin, O., Zoric, G., Engström, A., Reponen, E. Video Interaction: A Research Agenda. In *Personal and Ubiquitous Computing*, vol. 18 (2014). 685-692.

[14] Periscope. http://www.periscope.tv (24.05.2015)

[15] Boddhu, S. Williams, R., Wasser, E., Kode, N. Increasing situational awareness using smartphones. In *Proceedings of the SPIE*, vol. 8389 (2012).

[16] Sa, M., Shamma, D., Churchill, E. Live Mobile Collaboration for Video Production. In *Personal and Ubiquitous Computing*, vol. 18 (2014). 693-707.

[17] Procyk, J., Neustaedter, C., Pang, C., Tang, A., Judge, T. Exploring video streaming in public settings: shared geocaching over distance using mobile video chat. In *CHI '14* (2014). 2163-2172.

[18] Montané-Jiménez, L., Benítez-Guerrero, E., Mezura-Godoy, C. Towards a Context-Aware Framework for Improving Collaboration of Users in Groupware Systems. In *EAI Endorsed Transactions on Context-aware Systems and Applications*, vol. 1 (2014).

[19] Jorvid, N. Speaking Symbols: A semiotic analysis of the Smart Ping system in League of Legends. Uppsala Universitet (2014).

[20] Boix, E., Carreton, A., Scholliers, C., Van Cutsem, T. De Meuter, W., D'Hondt, T. Flocks: Enabling Dynamic Group Interactions in Mobile Social Networking Applications. In *SAC '11* (2011). 425-432.

[21] de Freitas, A., Dey, A. The Group Context Framework: An Extensible Toolkit for Opportunistic Grouping and Collaboration. In *CSCW '15* (2015). 1602-1611.

[22] Jain, P., Manweiler, J., Archarya, A., Beaty, K. FOCUS: Clustering crowdsourced Videos by Line-of-Sight. In *SenSys '13* (2013).

[23] OccupyStreams. http://occupystreams.org/ (17.09.2014).

[24] Web Realidade. http://www.webrealidade.org/ (23.07.2014).

[25] Engström, A., Perry, M., Juhlin, O. Amateur vision and recreational orientation: creating live video together. In *CSCW '12* (2012). 651-660.

[26] Engström, A., Zoric, G., Juhlin, O. The Mobile Vision Mixer: A mobile network based live video broadcasting system in your mobile phone. In MUM '12 (2012).

[27] XMPP. http://xmpp.org/ (02.05.2015)

[28] Smack API. http://www.igniterealtime.org/projects/smack/ (02.05.2015)