Towards a Wake-up and Synchronization Mechanism for Multiscreen Applications using iBeacon

Louay Bassbouss, Görkem Güçlü and Stephan Steglich
Fraunhofer FOKUS, Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany

Keywords: Multiscreen, Companion Screen, Application Wake-up, Synchronization, iBeacon, Bluetooth Low Energy.

Abstract: TV sets and companion devices (Smartphones, Tablets, etc.) have outgrown their original purpose and are now playing together an important role to offer the best user experience on multiple screens. However, the collaboration between TV and companion applications faces challenges that go beyond traditional single screen applications. These range from discovery, wake-up and pairing of devices, to application launch, communication, synchronization and adaptation to target device and screen size. In this position paper, we will limit ourselves to two of these aspects and introduce an idea for a new wake-up and synchronization mechanism for Multiscreen applications using iBeacon technology.

1 INTRODUCTION

Almost every modern connected TV can be controlled with a smartphone app provided by the manufacturer. These apps range from simple remote control replacements, to fully featured media centers including PVR programming, interactive EPGs and video streaming from the TV to the smartphone. Furthermore, new generation of companion devices provide connectivity to large displays and give mobile applications the ability to mirror or extend the small screen of the mobile device.

However, today's standard application models are focused on single devices and screens. Multiscreen applications face new challenges such as discovery of devices and services, launch of applications on remote devices, synchronization of application state across devices as well as timeline synchronization of multiple media streams, application to application communication, etc. New application development paradigms, concepts, protocols and technologies addressing these challenges are getting mature. However, there are still interoperability issues between different devices and platforms: Android (Google - Android, 2014) provides native support for screen mirroring using Miracast (Wi-Fi Alliance, 2014) protocol to any Miracast compliant receiver compared to iOS (Apple - iOS, 2014) that offers the same functionality using AirPlay (Apple - AirPlay, 2014) only on AirPlay licensed devices, such as AppleTV (Apple - Apple TV, 2014). There are also different limitations for applications running on different platforms: Android applications are able to run services in the background compared to iOS where this feature is limited to specific applications like Music, Navigation, etc. It is easier to interact with applications running in the background on Android systems than on iOS. The question is, why we need to interact with background applications on companion devices? To answer this question, let us consider the following scenario:

Alice and Bob are watching a Formula 1 race on their TV. The broadcaster offers a service which allows users watch the race from different camera perspectives, where the TV displays the perspective of the main camera and a companion device may be used to display the perspective of the on-board camera of a selected driver (user can select which driver). During commercial breaks, the broadcast stream on the TV shows advertisements for a certain brand of footwear. At the same time, the on-board camera stream will be interrupted on each companion device and a companion app related to the advertisement displayed on the TV will be launched. Through this app, the user can for example chose to bookmark products displayed in the commercial break that he is interested in buying later and may even get a discount. After the commercial break ends, the main perspective of the Formula 1 race continues on the TV and the on-board camera perspective continues on the companion devices.

The different steps of the use case are highlighted in figure 1. There are of course other relevant scenarios that require collaboration between TV and companion devices, but we will consider this scenario to examine the problem statement and identify the requirements on TV and companion devices.

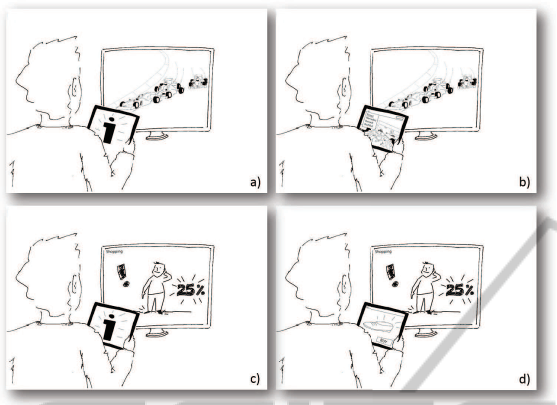


Figure 1: Use Case.

We can identify two main requirements from the scenario described above: (REQ1) wake-up and launch of companion applications related to broadcast services on the TV and (REQ2) Synchronization of content between TV and companion devices. There are other important multiscreen requirements that can be identified for this use case but in this paper we limit ourselves to the wake-up and synchronization aspects. The rest of this paper is structured as follows: Section 2 provides a short introduction to the fundamental state of the art technologies and related works required to understand the discussion of the proposed approach in this work. In section 3 we will introduce the concept of a new approach for wake-up and synchronization of companion applications using iBeacon technology. Section 4 summarizes the work with a conclusion and future prospects for a prof-of-concept implementation.

2 TECHNICAL DISCUSSION AND STATE OF THE ART

This section collects and weights prior work related to the concept we will introduce in this paper. This includes works which has been established during standardization efforts, as well as proprietary technology currently found in products on the end-user market. We will differ between application Wake-up and Launch on TV and on companion devices. The traditional way for launching applications on smart TVs is by using the TV remote control or a special mobile application provided by TV manufacturer. To launch a TV application that collaborates with an application running on companion device, e.g. YouTube application on TV for playing videos and YouTube mobile application to search and control videos, additional steps are needed. Both applications need to be connected to the same session in order to know about each other and establish a communication channel. PIN or QR code technologies can help here. The application running on the TV displays a PIN or QR code. The user of the companion device can either scan the QR Code using a QR scanner App or enter the PIN code manually in the companion application and join the session. To reduce the number of steps needed to connect both apps to each other, Network Service Discovery (NSD) technologies like SSDP (Contributing Members of the UPnP Forum, 2008) or Bonjour (Apple - Bonjour, 2014) can help. The Discovery and Launch Protocol DIAL (Netflix, 2014) is especially made for this purpose. TV sets that support this protocol implement the DIAL server functionality and advertise a DIAL service in the local network using SSDP. Companion devices implement the DIAL client functionality and discover DIAL devices available in the network. Once a DIAL device is found, the companion device connects to it and uses its service interface to launch a specific app. During Launch, the companion application can pass some parameters to the TV application such as session ID, video URL, etc.

In recent years, Hybrid TV technologies like Hybrid Broadband Broadcast TV (HbbTV) (HbbTV association, 2014), a major new pan-European initiative aimed at harmonizing the broadcast and broadband delivery of entertainment to the end consumer through connected TVs and set-top boxes, and Hybridcast (IPTV Forum Japan, 2014), a Japanese Hybrid TV standard that enables the provision of flexible and extensible new services that take advantage of the characteristics of broadcasting and telecommunications, are emerging. Involving companion screens within Hybrid TV sessions opens the door to a new kind of use cases such as the once described in the previous section and offers a replacement of traditional remote control that is not suitable for the interaction with such smart TV services. The key to successful integration of TV services and companion devices is the seamless and user friendly wake-up and synchronization of companion applications. Many Hybrid TV services are using the PIN or QR code technology for this purpose to ensure cross-platform support since these technologies are not dependent from platform specific APIs. On the other side, studies showed that these technologies are more suitable for printed material and not for the digital domain from user experience perspective (eMarketer, 2014).

Technically, The DIAL technology can be used for launching companion applications by implementing the DIAL server functionality on companion devices and client functionality on TV, but this will raise additional security and battery lifetime issues on companion devices. Furthermore, this requires companion devices to run services in the background which is not possible on some mobile platforms like iOS as we discussed in the previous section. In addition, the flow for remotely launching companion applications is not the same as for launching TV applications from the end-user perspective. Bringing an application on a companion device to the foreground without asking the user while he interacts with another application is a bad user experience. Most mobile platforms allow this feature only in relation with a user interaction e.g. the user starts a new application by clicking on a button in the current application or by clicking on a notification. On iOS, there are two types of notifications: local and remote notifications. The end-user don't see any difference between them, they differ only in the way how they are triggered: local notifications are triggered by applications running in the background on the same device while remote notifications are triggered and sent by Apple Push Servers. This means, if an application is not running at all (neither in the foreground nor in the background), the user can be notified only through remote notifications. One option to wake-up and launch a not running iOS application in the background is by using iBeacon (Apple - iBeacon, 2014), a new technology that extends Location Services in iOS7. iBeacon uses a Bluetooth Low Energy (BLE) (Bluetooth SIG, 2014) signal which iOS devices detect. Any device supporting BLE can be turned into an iBeacon transmitter and alert applications on iOS devices nearby. iBeacon transmitters are in general tiny and cheap sensors that can run up to 2+ years with a single coin battery depending on how frequent they broadcast information around. Main usage area of iBeacon are location based services: iOS devices alert apps when the user approaches or leaves a region with an iBeacon. While a beacon is in range of an iOS device, apps can also monitor for the relative distance to the beacon. If the application was not running while the user crosses (enters or leaves) a Beacon region, the iOS device wakes up the application and launches it in the background only for 10 seconds. In this time frame the application can react to the changes in the user position and may request to show a local notification, through which the user can bring the application to the foreground.

3 CONCEPT

Based on the technical discussion in previous section, we will propose some ideas for a user friendly remote launch mechanism of TV companion applications using iBeacon and Notification technologies. We will limit ourselves in this work to the iOS platform and follow its application development guidelines to ensure best user experience, nevertheless the concept can be easily adapted to companion devices with BLE support running other platforms like Android. Unlike on iOS, the only platform that provides native iBeacon support, the iBeacon functionality needs to be implemented on application level for the other platforms.

As mentioned in previous section, the iBeacon protocol uses BLE signals to transmit information in a specific frequency e.g. each second. An iBeacon message consists besides the BLE packet headers and Apple's static prefix of the following values (Apple - iBeacon for Developers, 2014):

Proximity UUID: A 128-bit value that uniquely identifies one or more beacons as a certain type or from a certain organization.

Major: A 16-bit unsigned integer that can be used to group related beacons that have the same proximity UUID.

Minor: A 16-bit unsigned integer that differentiates beacons with the same proximity UUID and major value.

iOS applications can use the iBeacon API available in iOS7 for registering a Beacon region using the proximity UUID, major and minor parameters described above. If a device crosses the boundaries of a registered beacon region, the application will be notified on entering or leaving that region. The proximity UUID is mandatory for registering a Beacon region while major and minor are optional. Application Provider needs to choose a value for the proximity UUID and use it in all beacons as well as in the iOS application. This means, the proximity UUID is in general a fix value in the context of a specific application or organization. Major and minor values can be used to differ between different locations or places for the same application or organization.

iBeacon seems to be a compromising technology not only for location based services, but also for launching companion applications in a multiscreen environment if new generation of TV sets (also set-top-boxes, TV dongles, etc.) are equipped with BLE sensors and act as beacon transmitters. Main advantage of this approach is that the TV will able to wake-up companion applications only on devices belonging to users sitting in front of the TV, it doesn't matter if they are connected to local or mobile carrier network. Unlike the traditional usage of iBeacon where the proximity UUID is fix and known for a specific application, a more dynamic behavior by using different values for proximity UUID on different TV sets is required in the multiscreen domain: If the TV manufacturer uses a unique proximity UUID, the companion application will be always notified when the user crosses the beacon region of any TV from the same manufacturer. In case TV sets transmit different proximity UUIDs, it will be possible to notify companion applications associated with a specific TV. Furthermore, it is possible for a companion application to subscribe to TV's different beacon regions at the same time and therefore to get notified by different TV sets, e.g. if the user has more than one TV at home. Figure 2 shows an example with two TV sets that transmit different proximity UUIDs. *Companion Device2* is registered for both *proximity UUID1* and *proximity UUID2* and can be notified from both *TV1* and *TV2*. The other two companion devices can be only notified from one TV set.

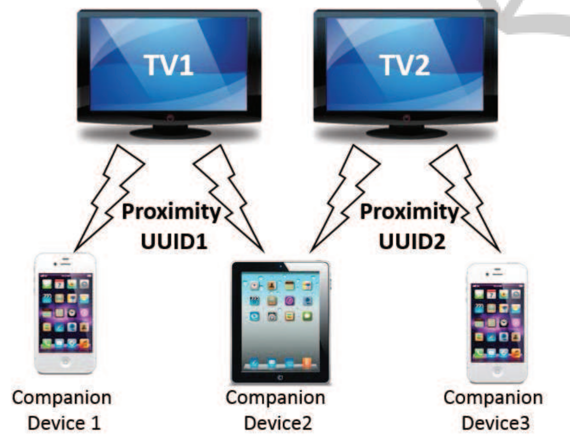


Figure 2: Example with two TV sets and three companion devices.

Since there is no unique and known proximity UUID to be used in the TV companion application, we need a mechanism to generate and exchange proximity UUIDs between the TV and the companion application. UUIDs can be randomly generated and stored on the TV without any user interaction. The probability of collision with UUIDs used in other applications is almost zero since proximity UUIDs are 128-bit long. The best way to exchange the generated UUID is during first connection (Setup phase) of the companion application with the TV. Figure 3 illustrates the steps needed for creation and exchange of proximity UUID between TV and companion application.

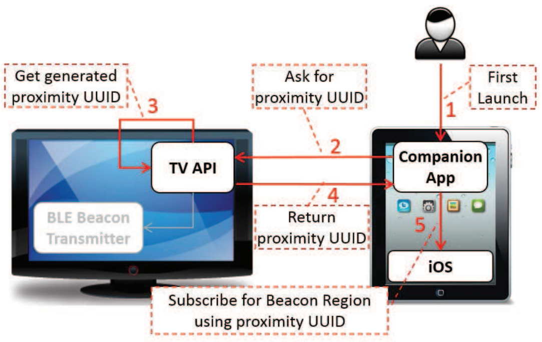
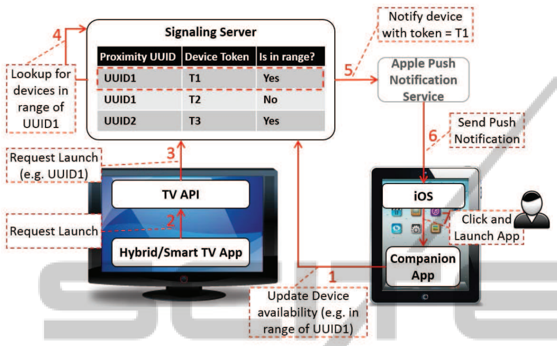


Figure 3: Creation and Exchange of proximity UUID.

From now on, each time the user turns on the TV or enters the TV's beacon region (e.g. living room) while the TV is on, the companion application will be woken up and launched in the background for around 10 seconds (iOS limitation). Same applies if the user turns off the TV or leaves the TV's beacon region. In both cases, the companion application connects to a signaling server (e.g. maintained by the TV manufacturer) when running in the background and requests to update its availability in the TV's beacon region by sending the proximity UUID and the device token (we suppose that the companion application already requested a device token from Apple Push Notification Service). As depicted in figure 4, the signaling server maintains a table for device availability and offers a lookup function to find devices (identified by the device token) in range of a specific beacon (identified by the proximity UUID). On the other side, If a TV Application (Hybrid or Smart Application) provides multiscreen support and needs to launch a companion application, it uses a special TV API for that purpose (Figure 4 - step 2). The TV sends its proximity UUID and other application specific information to the Signaling Server (Figure 4 - step 3). The signaling Server looks into the table for all devices in range of the beacon with the received UUID (Figure 4 - step 4) and sends a request to the Apple Push Notification Service (APNs) (Apple Developer - Local and Push Notifications for Developers, 2014) using the tokens of devices in range from the previous step. The APNs sends push notifications containing all information necessary to launch the companion application to each found device (Figure 4 - step 6). If the user clicks on the Push notification, the companion application will be launched into the foreground and the notification data will be passed to it (Figure 4 - step 7). Apple's Push Notification Service for remote push notifications is necessary in this scenario, because local notifications are only possible, when the app is running in the background at that moment. Though, apps are woken up through iBeacons, they are only running for ten seconds and get terminated by the OS afterwards. Moreover, the OS will only wake up apps for entering and exiting a region of a beacon. As a result, we can not assume the companion app is running at that moment and is available to send a local notification. Therefore, it is necessary to use Apple's Push Notification Service for remote push notifications.


Signaling Server
Notify device with token = T1
4
Lookup for devices in range of UUID1
5
Apple Push Notification Service
6
Send Push Notification
Request Launch (e.g. UUID1)
3
TV API
2
Request Launch
Hybrid/Smart TV App
iOS
7
Click and Launch App
Companion App
1
Update Device availability (e.g. in range of UUID1)

Proximity UUID	Device Token	Is in range?
UUID1	T1	Yes
UUID1	T2	No
UUID2	T3	Yes

Figure 4: Launch a Companion Application from a TV Application.

Figure 4 shows the flow for notifying and launching the companion application provided by the TV manufacturer from a hybrid or smart TV application provided by a broadcaster or a third party provider. However, our goal is to launch the companion application related to the TV application and not the manufacturer companion application. There are different options to achieve this goal depending from the kind of the companion application to launch, if it is a hosted web application or a native iOS application. We will focus in this work on hosted web applications and consider native iOS companion applications in future works. As mentioned above, The TV application passes in the *Request Launch* step (Figure 4 - step 2) information about the application to launch on the companion device. This information could include a URL of the hosted web application to launch and will be passed to the companion device through all steps depicted in figure 4 until the user clicks on the push notification: The TV companion application will be launched in the foreground and can retrieve the URL of the hosted companion web application from the launch information passed to it. Finally, the TV companion application opens the hosted companion application in a Web View (*UIWebView*), a kind integrated web browser for displaying web content in iOS applications. The TV application and the hosted companion web application can collaborate and synchronize content with each others by using an application to application (App2App) communication mechanism which is not in scope of this work (A widespread mechanism for communication in the multiscreen domain is to connect the TV and companion web application to a messaging server that proxies the communication between entities in same session. Session ID could be passed to the companion web application during launch as a URL query parameter).

Besides application wake-up and launch, iBeacons could also help in synchronizing playback of multiple media streams as in the use case described in first section for synchronizing streams from different camera angles on TV and companion device. The major and minor values can be used for this purpose e.g. major value to group beacons related to the same application on TV and companion device and minor value can be controlled by the TV application e.g. to send identifiers indicating the current position in the TV stream. While the TV companion application runs a hosted companion web application in the foreground, it listens for beacons with the same major value as for the related TV application and passes minor values to the application using an API provided by the Web View. Figure 5 summarizes the synchronization steps between TV and companion application.

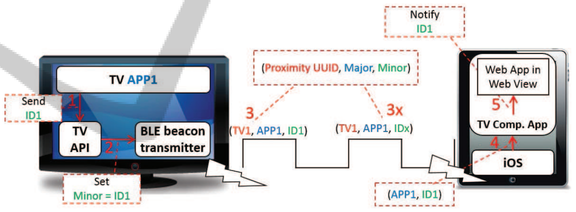


Figure 5: Synchronization between TV and Companion Web Application using beacon's major and minor values.

4 CONCLUSION

We introduced in this paper a Wake-up and Synchronization Mechanism for Multiscreen Applications on TV and companion devices using iBeacon technology. We already started with a proof-of-concept implementation using iPad as replacement for TV and any iOS device supporting iBeacon as companion device. The prototype was demonstrated at the 4th Media Web Symposium organized by Fraunhofer FOKUS in Berlin (Fraunhofer FOKUS Media Web Symposium , 2014). In next steps we will take into consideration other important aspects that could influence the success of work introduced in this paper especially usability, privacy, security and performance. On prototyping level we plan to implement the TV part on an Android HDMI Dongle with BLE and DVB-T/S/C support in order to proof our concept with real Hybrid TV (HbbTV) applications.

REFERENCES

Apple - AirPlay (2014). Apple - airplay. [Online; accessed 4-June-2014].

Apple - Apple TV (2014). Apple - apple tv - hd itunes content and more on your tv. [Online; accessed 4-June-2014].

Apple - Bonjour (2014). Bonjour - apple support. [Online; accessed 4-June-2014].

Apple - iBeacon (2014). ios: Understanding ibeacon. [Online; accessed 4-June-2014].

Apple - iBeacon for Developers (2014). ibeacon for developers - apple developer. [Online; accessed 4-June-2014].

Apple - iOS (2014). Apple - ios7. [Online; accessed 4-June-2014].

Apple Developer - Local and Push Notifications for Developers (2014). Local and push notifications. [Online; accessed 4-June-2014].

Bluetooth SIG (2014). Bluetooth smart technology: Powering the internet of things. [Online; accessed 4-June-2014].

Contributing Members of the UPnP Forum (2008). Upnp device architecture 1.1. [Section 1].

eMarketer (2014). Us ahead of western europe in qr code usage. consumers most familiar with qr codes on magazines, printed materials. [Online; accessed 4-June-2014].

Fraunhofer FOKUS Media Web Symposium (2014). 4th fokus media web symposium. [Online; accessed 4-June-2014].

Google - Android (2014). Android. [Online; accessed 4-June-2014].

HbbTV association (2014). Hbbtv. [Online; accessed 4-June-2014].

IPTV Forum Japan (2014). Hybridcast. [Online; accessed 4-June-2014].

Netflix (2014). Discovery and launch protocol specification, version 1.7.

Wi-Fi Alliance (2014). Wi-fi certified miracast. [Online; accessed 4-June-2014].