

# Opportunistic Communication for Multiplayer Mobile Gaming: Lessons Learned from PhotoShoot

Roelof Kemp, Nicholas Palmer, Thilo Kielmann and Henri Bal  
Vrije Universiteit  
De Boelelaan 1081A  
Amsterdam, The Netherlands  
{rkemp, palmer, kielmann, bal}@cs.vu.nl

## ABSTRACT

In this paper we describe how a mobile multiplayer game benefits from opportunistic communication.

## Categories and Subject Descriptors

C.2.1 [Computer-Communication Networks]: Network Architecture and Design—*Network Communications*

## General Terms

Experimentation

## 1. INTRODUCTION

In the last few years we have seen the introduction of mobile phones with large touchscreens, various sensors, built-in cameras, and support for multiple communication protocols, such as Bluetooth, WiFi and 3G. These so-called *smartphones* offer functionality far beyond that of traditional mobile phones, which only allow for making phone calls and sending text messages.

With a continuing increase in market share, smartphones are likely to largely replace traditional phones in the coming years and have already been hailed as the 'next wave in computing' [12]. Initial steps have been made in new application areas such as augmented reality [11], social networking [9], and context-aware applications [4] exploiting various new features of smartphones. The new communication possibilities have led to many novel *phone-to-Internet* applications like Internet browsers and client applications for various cloud services, such as calendar, email and navigation services, as well as a few *phone-to-phone* applications, such as multiplayer games.

Current multiplayer games for smartphones tend to use the phone-to-Internet type of communication with a *fixed centralized server*. Such a central server, located at a fixed address, maintains the game state and routes communication to participating players. Next to the disadvantage of the cost of maintaining such a server, the server-side software of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*MobiOpp* '10, February 22-23, 2010, Pisa, Italy.

Copyright 2010 ACM 978-1-60558-925-1/10/02 ...\$10.00.



**Figure 1: In PhotoShoot one shoots virtual bullets at a real opponent.**

the game needs to be designed carefully with scalability in mind, since the server might become heavily used for popular games. Furthermore, the game stops if the network gets partitioned with the phones being in another partition than the server.

In this paper we describe our ongoing research on how our Ibis communication middleware can simplify the development of novel multiplayer games for smartphones by offering true phone-to-phone *ad hoc communication* without the need for a fixed centralized server. Furthermore, we present the lessons we have learned from building the application 'PhotoShoot - The Duel', an innovative augmented reality multiplayer game, on top of the Ibis middleware.

## 2. PHOTOSHOOT

Inspired by Google's second Android Developer Challenge [1], we developed *PhotoShoot*<sup>1</sup>, an innovative and original distributed augmented reality game that is competitive with respect to the challenge and fits in our ongoing research on smartphone middleware in the Ibis project. The application provides us with new insights into the applicability of the Ibis middleware for smartphones: it is the first Ibis application that runs distributed on multiple phones.

PhotoShoot augments reality by allowing two people to fight a *virtual* duel in the *real* world, using their smartphones as weapons, photographs as bullets, and each other as opponents (see Figure 1). The players' smartphones act as referees throughout the process of the duel. They impose the following rules on both persons:

<sup>1</sup>PhotoShoot made it to the final of the ADC-2 and finished 6th place in the category 'Action Games'.

- The duel starts at the same exact time.
- The players take a fixed number of steps in opposite direction.
- Only after the steps have been taken, the players are allowed to turn around.
- The duel itself lasts 60 seconds.
- The maximum number of shots per player is 6.
- The first player that shoots a picture of the opponent's face will win.

PhotoShoot runs on Android, a new, open-source operating system designed to be used on mobile phones, developed by the Open Handset Alliance [6], in which Google is one of the key participants. Android provides a rich set of application programming interfaces (APIs), that can be accessed through the Java programming language. In contrast with the commonly used mobile version of Java (JME [2]) the Android APIs are nearly complete with respect to standard Java. In addition, Android provides smartphone specific APIs to access sensor information, such as the geolocation of the phone.

We use Android's additional sensor APIs to detect *gestures*, movements of the person holding a phone. Android has no direct high-level APIs for gesture detection, but it supports two types of sensors to retrieve movement-related information: the accelerometer and the digital compass. By analyzing the sensor information, we can detect the gestures the user makes. For PhotoShoot we use the sensors to detect steps and turns of the player during the initial phase of the duel.

Furthermore, we make use of Android's support of face detection on camera images. Since the face detection algorithm is a computation and memory-intensive operation, we can only run the algorithm over a small part of the image. In the case of PhotoShoot, we limited the image size by only detecting faces in the area covered by the crosshairs. Detection on larger images requires advanced techniques, such as Ibis-based *cyber foraging* as described in [3], to offload heavy weight computation to other compute elements.

## Technical Requirements

PhotoShoot needs two Android-powered devices that share a network (i.e. WiFi, 3G or Bluetooth) to run.

## 3. IBIS MIDDLEWARE

Now that we have seen how we can use the Android operating system to implement the *intra* phone requirements, we will shift focus to the *inter* phone requirements regarding the communication between the phones.

The Android operating system offers communication interfaces in the form of standard socket connections for WiFi and 3G, as well as Bluetooth communication channels, which all can be directly accessed by multiplayer games. Despite the availability of these interfaces it is a difficult task to write an application that supports multiple networking interfaces in different networking scenarios. Therefore, many existing multiplayer games on smartphones are limited to a specific networking interface and do not fully utilize the phone's capabilities; they rarely support WiFi, 3G and Bluetooth together. Furthermore, multiplayer games tend to

use a centralized server for communication setup and device pairing, thereby requiring a permanent Internet connection to the server and being vulnerable to network partitioning. The centralized server needs to be highly scalable, to handle many connections at a time, which makes the development of multiplayer games even harder.

In order to ease the writing of multiplayer smartphone games, we extend the Ibis middleware [10] to provide a single stable communication interface, which will bind to any available network. Originally designed for the domain of High-Performance Distributed Computing (HPDC), the Ibis middleware has proved to also be useful for Mobile Computing [3, 7]. The key communication library of Ibis is the Ibis Portability Layer (IPL), which offers an API for unidirectional communication channels between arbitrary numbers of SendPorts and ReceivePorts. Applications that use the IPL communication need to connect to an Ibis Server which includes a registry that tracks available resources (devices). The Ibis Server is a light weight process and can be run on the phone itself, thereby making the Ibis middleware ideally suitable for opportunistic adhoc communication.

In the following sections we will discuss how the IPL binds to different networks, how we set up IPL communication and how we solve the problem of device pairing.

### 3.1 Adhoc Network Binding

Both the Ibis Server and the IPL library have a single interface to the application, but can be bound to multiple implementations that each support a particular networking interface, such as Bluetooth or TCP. The appropriate implementation will be selected adhoc at runtime. Thus, any IPL application will run on top of different networks without the need of changing the code, reconfiguring or recompiling. Among the implementations are a SmartSockets [5] implementation, which supports WiFi and 3G connections in situations where normal socket connections would fail (Network Address Translation, firewalls, multi-homing), a TCP implementation and an experimental Bluetooth implementation.

### 3.2 Adhoc Communication Setup

In order to initiate communication, all phones have to bind to the registry service running in the Ibis Server. In the HPDC domain, this Ibis Server is started before the distributed application itself and the address of the server will be given to any instance of the distributed application.

In the smartphone domain, however, it is unlikely that a single actor controls all the participating devices including an additional device to run an Ibis Server. Therefore, the communication setup is adapted to support *adhoc* setup. In adhoc setup, each application has the potential to start up a *local* Ibis Server on the phone itself or to connect to an existing Ibis Server on another device (see Figure 3), thereby avoiding the need for a costly *central* server. Because each duel creates its own temporary local server, there is no need to scale the central server in case the game becomes popular. Another disadvantage of centralized applications is that they are limited to environments with a permanent connection to the central server on the Internet, a situation that can be infeasible due to limited coverage as well as undesirable because of the costs it involves. IPL applications, however, can take advantage of local adhoc networks such as wireless adhoc networks and Bluetooth piconets.

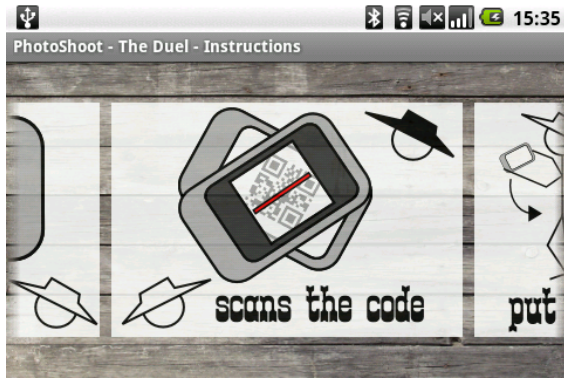


Figure 2: Device pairing in PhotoShoot. A screen capture of the instructions for the user.

### 3.3 Adhoc Pairing

In order to *pair* (i.e. establish an initial connection) all participants to the phone hosting the Ibis Server, the other phones need to discover the address of the Ibis Server. In the HPDC domain no discovery is needed, since the Ibis Server address is considered to be well known before any of the applications start. However, in adhoc settings this information is unknown and needs to be discovered by other phones. We therefore extended Ibis to use a side channel to exchange the Ibis Server address. We use a visual side channel and encode the Ibis Server address in a two dimensional barcode, known as a QR-code [8]. The phone hosting the Ibis Server displays this QR-code and using the camera sensor other phones scan this address (see Figure 2 and 3).

### 3.4 Connection Changes

After running initial experiments during a 2 day career event for high school students in The Netherlands, where approximately 200 visiting 16-18 year old students played PhotoShoot, we found that it is essential for a distributed application to be prepared for unforeseen connection interrupts. Due to both software and physical causes network connection may at any time be suddenly interrupted. Although it is not always possible to recover from network interruptions, distributed applications need to be prepared to handle them. Since the Ibis middleware is designed for highly dynamic environments where nodes join and leave continuously, it provides the mechanisms to handle connection changes. Although we left recovering of network failures as future work, we added connection change support to PhotoShoot so that it terminates gracefully when one of the phones leaves the application.

## 4. CONCLUSIONS

PhotoShoot demonstrates how the Ibis middleware can be used for applications that require opportunistic inter phone communication. We proposed a new type of setup for distributed IPL applications on smartphones that is suitable for adhoc environments. Using adhoc communication setup each instance of the distributed application has the potential to host an Ibis Server. To discover the address of a remote Ibis Server to pair with it, we added an innovative device pairing system that shares the address of the Ibis Server in a QR-code. This setup has advantages over the current fixed

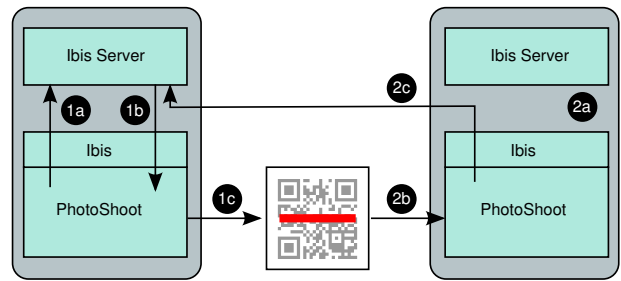


Figure 3: Adhoc Communication Setup. The phone that starts the duel, starts a local Ibis Server (1a), retrieves its address (1b) and displays it in a QR-code (1c). The accepting phone starts no local Ibis Server (2a), but scans the QR-code (2b) and connects to the remote Ibis Server (2c).

centralized server approach in terms of cost and scalability. Furthermore, it is less vulnerable to network partitioning, since it does not require a permanent Internet connection.

The combination of the Android operating system together with the Ibis middleware forms a solid base which simplifies the development of innovative distributed smartphone applications, such as the augmented reality multi-player game PhotoShoot.

## 5. REFERENCES

- [1] Android Developer Challenge 2. <http://code.google.com/android/adc>.
- [2] Java Micro Edition. <http://java.sun.com/javame/>.
- [3] R. Kemp, N. Palmer, T. Kielmann, F. Seinstra, N. Drost, J. Maassen, and H. E. Bal. eyeDentify: Multimedia Cyber Foraging from a Smartphone. In *IEEE International Symposium on Multimedia*, 2009.
- [4] Two forty four a.m. website. <http://www.twofortyfouram.com/>.
- [5] J. Maassen and H. E. Bal. Smartsockets: solving the connectivity problems in grid computing. In *HPDC '07: Proc. of the 16th int. symposium on High performance distributed computing*, pages 1–10. ACM, 2007.
- [6] Open Handset Alliance. <http://www.openhandsetalliance.com/>.
- [7] N. Palmer, R. Kemp, T. Kielmann, and H. E. Bal. Ibis for mobility: solving challenges of mobile computing using grid techniques. In *HotMobile '09: Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, pages 1–6. ACM, 2009.
- [8] Denso Wave's QR website. <http://www.denso-wave.com/qrcode/index-e.html>.
- [9] Twidroid website. <http://twidroid.com>.
- [10] R. V. van Nieuwpoort, J. Maassen, G. Wrzesińska, R. F. H. Hofman, C. J. H. Jacobs, T. Kielmann, and H. E. Bal. Ibis: a flexible and efficient java-based grid programming environment. *Concurr. Comput. : Pract. Exper.*, 17(7-8):1079–1107, 2005.
- [11] Mobilizy website. <http://www.mobilizy.com/wikitude.php>.
- [12] P. Zheng and L. Ni. *Smart Phone and Next Generation Mobile Computing*. Morgan Kaufmann, 2006.