A Multi-tier Client-Server Project Employing Mobile Clients

Chris McDonald
School of Computer Science and Software Engineering
The University of Western Australia
Crawley, Western Australia, 6009
chris.mcdonald@uwa.edu.au

Abstract

The current generation of Computer Science students are far more likely to engage with computer networking through their own mobile, wireless devices than they are using wired, desktop computers. Traditional approaches to teaching computer networking evolved when the Internet was composed of fixed wired infrastructure, and this historical background still forms most of the material in contemporary textbooks on computer networking. Today's students have strong expectations that their computer networking units will have a significant focus on the networking devices and applications that they use daily - increasingly mobile and wireless.

This paper describes a client-server networking project which requires students to design, implement, test, and analyse, a client-server software architecture, using both desktop computers, and handheld, mobile, wireless devices. The application domain of the project is location prediction using only WiFi beacon frames – an application familiar to most students, but one about which they initially had little curiosity. The paper then reflects on the many lessons learnt from this project, both from the perspective of the students and the professor.

Keywords: Mobile and Wireless Networks, Location Prediction, Computer Science Education

1 Introduction

It can be argued that computer networking continues to be the field of computing making the greatest impact on our daily lives, with the Internet becoming the ubiquitous carrier of global applications such as user-contributed videos and music, instant messaging, location-aware mapping and applications, and the myriad of competing and often contradictory information sources. Our students' almost insatiable desire for access to the Internet often pushes aside the curiosity to understand the technical aspects of how it all works, and even why it all works. Burd et al. (2012) present extensive arguments as to why the rise, role, and popularity of mobile computing throughout society should be strongly supported by a re-examination and update of relevant areas of the Computer Science curriculum.

Contemporary university students, collectively described as the M-generation, Generation-Y, and Millennials, are more likely, or wish, to access many Internet services using handheld and mobile devices, including laptop computers, smartphones, and tablet computers, as reported by Junco & Mastrodicasa (2007), Oblinger & Oblinger (2005).

The rapid growth in mobile and wireless computing, and its emerging role in Computer Science Education, is easily witnessed by the number of recent publications in the literature. Many of these publications, particularly those form as "long ago" as 2008, have reported on the application of using mobile devices to teach design and human-centric computing, often focusing on the challenges of the devices' small form-factor. More recent papers have described the introduction of mobile phone application development into Computer Science curricula, both at university and college level and, more significantly, into the general computing curricula at high-school and (the US) K-12 levels. Specifically, the use of mobile devices and sensors are often used in university and school Outreach Programmes, to motivate prospective university students. Many newer publications focus on the use of existing application development environments, mostly for the Android platform, but a few for Apple's iOS, and often employ mobile games development or use of the devices' inbuilt 3D, light, and sound sensors (Bayzick et al. 2013, Dabney et al. 2013).

More relevant to this paper, have been a number of publications that focus on the use of mobile and wireless devices *as general purpose computers*, and as vehicles to study established areas of Computer Science such as Operating Systems and Computer Security (Andrus & Nieh 2012, Riley 2012, Guo et al. 2013).

2 Motivation

For current students in Computer Science units the Internet's "last-mile" problem is being solved, not through traditional fibre or copper connections to the home, but by near-ubiquitous wireless access. Consequently, students undertaking units in computer networking today are expecting their units to contain, if not focus on, details of wireless and mobile networking, developed this century, and not to remain focused on the traditional, often inaccessible, wired backbone infrastructure and their protocols, developed in the 1970s.

While it remains necessary that Computer Science units explain the networking fundamentals of data corruption and loss, switched versus packet-based transmission, layered models, and the roles and responsibilities of numerous protocols at

all layers of the networking stack, the always-on, instant-connect nature of broadband connections and mobile telephony effectively pushes aside curiosity about the detailed technical infrastructure in place. Traditional topics, such as Shannon's communication theory, error correcting codes, and algorithms drawn from graph theory, can no longer hold pride of place in the networking classroom.

To maintain students' engagement and, more importantly, to increase their understanding of the design and implementation of modern computer networks, it is now necessary to convey many of the fundamental concepts by drawing upon mobile and wireless networking examples. This paper presents our recent successes with student projects developing a location prediction application using the wireless communication facilities of modern mobile devices, such as Apple iPods.

An increasing number of mobile handheld devices, described as smartphones, provide Internet connectivity through wireless ethernet (WiFi) and well as through cellular phone infrastructure (in Australia, 3G and 4G). Higher-end models also provide GPS receivers to provide accurate location information. However, the results that can be delivered by GPS alone fails or degrades in a number of environments, including urban canyons and the interior of buildings. This is precisely where WiFi can exhibit great advantage – in environments containing many WiFi access points, such as on a university campus. Most modern smartphones and their operating systems support hybrid approaches to localization, combining GPS signals, the known locations of fixed telephony infrastructure (mobile phone towers), and explicitly collected or crowd-sourced information including the MAC addresses and signal strengths of observed WiFi access-points.

Students are very familiar with the location prediction software provided by the operating systems, and third-party applications, on their smartphones. Depending on the types and strengths of signals available to the device, location determination generally employs the device's GPS receiver, or a combination of GPS and WiFi signals. Devices report results and changes when time or distance intervals are exceeded. Location determination using WiFi signals emitted from known WiFi access-points usually requires access to a very large database of pre-captured information, such as that provided by the mapping services of Google, Apple, and Skyhook Wireless. However, when used alone, location prediction using only WiFi often only claims an accuracy to 150 metres in urban areas, as prediction is obviously limited by the number and accuracy of samples taken.

Wikipedia (Wikipedia 2012*a*) describes geolocation as *"the identification of the real-world geographic location of an Internet-connected computer, mobile device, website visitor or other... Geolocation may refer to the practice of assessing the location, or to the actual assessed location, or to locational data."*

This paper reports on projects undertaken in successive years, by undergraduate students to form part of their assessment in their core computer networking unit. In these projects students implemented a form of geolocation by employing the WiFi signal information received by handheld mobile devices. While the problem domain, and its uses, are very familiar to our students, the projects have demonstrated areas of our teaching and of our students' knowledge that could do with improvement.

3 Background

The *Computer Networks* unit at The University of Western Australia is offered to students in their third undergraduate year, students undertaking bachelor degrees in Computer Science, Software Engineering, Electronic Engineering, and Business, and to students undertaking a "conversion" Masters by Coursework degree, whose first degree is frequently other than computing or information technology. While most undergraduate students have taken other systems-focused units, such as *Operating Systems* and *Security and Privacy*, most of the unit's Masters by Coursework students have arrived with very diverse backgrounds, which has caused many challenges in this unit. In recent years the unit has received enrolments of between 55 and 160 students.

The unit comprises one eighth of a year's work, and is assessed with a practical project contributing 40% and two examinations contributing 60%. In recent years, the practical projects described in this section, ran for 5 weeks each and were undertaken in self-selecting groups of two or three students.

The goal of each student project was to design, implement, and analyse a location-prediction application using a multi-tier client-server architecture. Students were to develop software forming part of a mobile application, to execute on Apple iPod Touch devices, and a simple server running on a desktop machine (typically running Linux) which communicates with a provided black-box server using a pre-defined protocol. As well as developing working software, students were also required to to analyse and report on the observed effectiveness of their software. The project was designed to improve students' understanding of the physical wireless environment, IEEE802.11 wireless protocols, and to assess their understanding of Transport Layer networking protocols, and their ability to analyse and report on their observations. While the project was not designed to teach skills in mobile architectures or the programming of mobile or embedded devices, the project also provided an initial exposure to these topics for most students.

In some schools, such a project may appear under-specified, as very little explicit information was provided as to how students should undertake their mobile and server application design and evaluation. Students had already completed four weekly closed laboratory sessions which had introduced them to the data-link layer, routing, and wireless network protocols. The *cnet* network simulator (McDonald 1991, 2009) had, in particular, encouraged them to experiment with the fundamentals of wireless signal transmission and reception, typical transmission distances, and the handling of frame collisions. However, the use of *cnet* for many years had clarified, in the author's mind, that too many students were perceiving the simulator, and its ease of use, to be an accurate reflection of programming and testing mobile applications. While simulation certainly still holds a deserved position in the teaching of computer networking, the emergence of mobile and wireless devices means that we should now offer our students a richer experience.

4 Project Methodology

The complete project was undertaken over five weeks, by teams of two or three students. The project employed a small number of distinct software applications and network protocols, and students'

solutions required two primary phases to achieve a successful outcome.

4.1 The provided *photoserver* software

The author developed a simple server, named the *photoserver*, to host and deliver GIF-format images that had been taken using a smartphone. For our project, about 300 equally-sized images from 60 locations were collected from a region of about 400x300 metres, near our Computer Science building. The latitude, longitude, and compass direction faced (when taken) of each image were manually recorded although, today, many digital cameras and smartphones can themselves provide this information.

Students were provided with details of how to contact the *photoserver*, including its IPv4 network address and TCP-based port with which it could be contacted. At the conclusion of the project, interested students were provided with the source code to the server (although similar implementations are readily available over the Internet). Student also received details of the simple TCP/IP text-based protocol used to communicate with the server. The protocol defined a small number of commands (verbs) understood by the *photoserver*, and its valid responses and error conditions. In essence, the commands and responses were similar to many existing text-based standards, such as for FTP or TELNET. Commands could request that the *photoserver* report the geographical region that it was "managing", and to request the downloading of images from specific locations (latitude, longitude, and heading) within the region.

While students received details of the *photoserver's* protocol, they were *not* provide with a pre-built client-side application programming interface (API), in any specific programming language, with which to access the *photoserver*. This enabled students to write their *geoserver* software (described later) in their choice of programming language, and required them to implement the necessary bidirectional communication using their language's Berkeley socket APIs and I/O functions or methods. Most students adopted their preferred language taught and used in their earlier units – Java, C, or Python in decreasing order of "popularity".

These design choices, in themselves, provided three important lessons to students – that clients and servers communicating over the Internet need not be written in the same programming language, that text-based protocols facilitate communication between disparate operating systems and hardware, and that text-based protocols are generally sufficiently efficient for such simple tasks. Because the *photoserver's* protocol was text-based, students could experiment with it using existing command-line programs, such as *netcat*, and then have their own programs mimic these actions by sending text-based commands, across a socket, and receiving and parsing responses.

The *photoserver* software was also deliberately placed behind a firewall that prevented it being accessed, directly, from the students' mobile devices. This enforced the project's requirement that students develop both a mobile device application and a desktop server application (described shortly). Sufficiently enthused students *could* have added more photographs from across the whole campus, but none of our students even asked why the chosen region was so limited – it is likely that none ventured outside of this region.

4.2 The students' mobile application

The second part of the project required students to walk around the region, capturing WiFi *beacon signals* from the university's WiFi network. Our university has an extensive WiFi network, covering nearly all of its campus, with about 50 of its 360 wireless access-points "visible" in the physical region of our project. The students were provided with 8GB Apple iPod Touches for the duration of the 5-week project, and with code for a mobile application to extend and run on the devices. The application's provided code captured signals from the device's wireless network interface, and managed a simple multi-screen interface; see Figure 1. The interface enabled the students to set the network address of their *geoserver* software, displayed a map of the campus on which locations of previously captured WiFi beacons could be identified, a screen to display images fetched from the *photoserver* (via their *geoserver*), and a text-based logging/debugging screen.

The iPod's mobile application was written by the author in Objective-C, and made calls to Apple's iOS library to provide the graphical interface and to gather information about captured WiFi beacon frames. Our students gain no direct experience with Objective-C throughout their degree, although some learn it through their personal interest in programming Apple iOS and OS-X devices. Learning a new programming language is not an objective for our *Computer Networks* unit, and so students were encouraged (permitted) to develop all of their own code on the iPods in ISO-C99, a language they *had* previously learnt in earlier years, and had also used extensively in a recent Operating Systems unit.

Students were provided with a standard *Makefile* to compile and link the provided Objective-C code and their own C code, to build the iPod application, and a simple *ssh/scp* command sequence to push their applications to the mobile devices over a USB connection. The number of students enrolled in the unit introduced a difficulty not initially anticipated – despite working in groups, almost every student wished to undertake some development of their team's mobile application. Our school could not provide access to sufficient Apple desktop machines to meet this demand, and so the author developed a network-based cross-compiler enabling students working on the school's Linux desktops to have their mobile applications compiled and linked on Apple hardware. Although the project required some investment in and commitment to Apple's iOS platform, it could similarly be developed and undertaken using Android on suitable devices.

The student written C code in the mobile application responded to button presses on the device's screen. In the first phase of the student's data collection, students scrolled the on-screen map to align crosshairs with their actual current location, and then pressed a button to provide their code with triples of the most recently captured WiFi information, including each access-point's MAC address, signal set identifier (SSID), and the relative signal strength of the arriving signal (its RSSI) value. In combination, this information forms a wireless *fingerprint* of the location where it was captured. The information was sufficient for the students' projects to determine the approximate physical location of their mobile device, and for that information to be further used to request a photographic image of the scene that should be observable from that location – a poor man's *Google StreetView*.

Figure 1: Sample screens from the students' mobile application.

Students employed a variety of strategies to process their captured data, with decisions taken dependent on their related choices for their *geoserver's* design. Nearly all student teams transmitted their captured data back to their *geoserver* as formatted ASCII text, often with one WiFi reading per line. It is unclear if students generally recognized the benefits of, and chose to use, a text-based format based on the early discussions in class sessions, or if they were simply mirroring the provided protocol between the *photoserver* and their *geoserver*. Most teams, initially, chose to employ TCP stream connections to their server software, and sent the captured information from a single location as a sequence of lines. Other teams packed the same information into a single UDP datagram, resulting in their need to also add sequence numbers and their tracking to their protocols. The challenges introduced by lost data while using UDP datagrams led a number of teams to adopt TCP streams, instead.

4.3 The students' *geoserver* software

The students' second task was to design and implement their *geoserver*, software sitting midway between the provided *photoserver* and their mobile application, and acting as a proxy for requests, results, and errors. As the *photoserver* was firewalled from our campus wireless network, and thus from the mobile applications running on mobile devices with known DHCP-allocated addresses, all communication had to, at least, pass through the *geoserver*s. Students' *geoserver*s ran on either desktop machines or on virtual servers in our Computer Science building. Ironically, during testing, students "in the field" often used their mobile phones to communicate (by phone) with other students controlling their team's *geoserver* application on the desktop machines.

The primary role of the *geoserver* was to determine the *apparent* location of the mobile device, based on the WiFi signal information that it was currently receiving from the mobile device. This concept of an *apparent* location was central to the project, but was initially misunderstood by many students. Although our campus is well covered by over 360 access-points, the project did not require knowledge of their actual location. Instead, the fingerprints collected during the project's first phase were used to determine each *access-point*'s apparent location – holding the location beneath the mobile interface's crosshairs as the ground-truth, and then predicting where each access-point's location was "around" this point.

With sufficient walking and sampling of the region, each access-point is observed several times, with its WiFi beacon frames arriving at each claimed location with a variety of signal strengths. At the end of the project's first phase, each team had used their captured data to determine an *apparent* location for each access-point, but each particular access-point's apparent location varied, depending on from where, and how many times, it was observed. Student teams employed a variety of algorithms, including self-described "intuitive" algorithms, to determine apparent locations from sampling locations. The most common of these were traditional triangulation and trilateration (Wikipedia 2012*b*,*c*). The first of these is the simpler, and can be easily derived anew from basic geometry – in fact one team claimed to have discovered it, and named it after themselves. Triangulation, although simple, does not take into account the strength of arriving signals and, thus, places all sampled points equidistant from the sample centre.

More complex is trilateration, which uses the arriving signal strength to represent a logarithmic inverse of the distance between the sampling centre and access-point; see Figure 2. As the signal strength of beacon frames from access-points dissipate according to the inverse-square law, trilateration provided more consistent results when predicting new locations with respect to previously visited ones.

Some student teams employed techniques that were reliant on them being able to find the wireless footprint most-similar to ones previously captured, and two teams even attempted to develop a detailed grid of the whole (assumed) environment, to a resolution of 5 metres. These latter two approaches often led to unsuccessful projects (ones not being able

to accurately predict current locations, or to deliver a photo from the predicted location), as the techniques were reliant on very repeatable observations. Such attempts were quite disappointing, as the students could have easily observed that, even while standing still, received signal strengths vary by amounts that would displace their predictions by as much as 30 metres. The clear inference is that students did not perform enough testing of their applications.

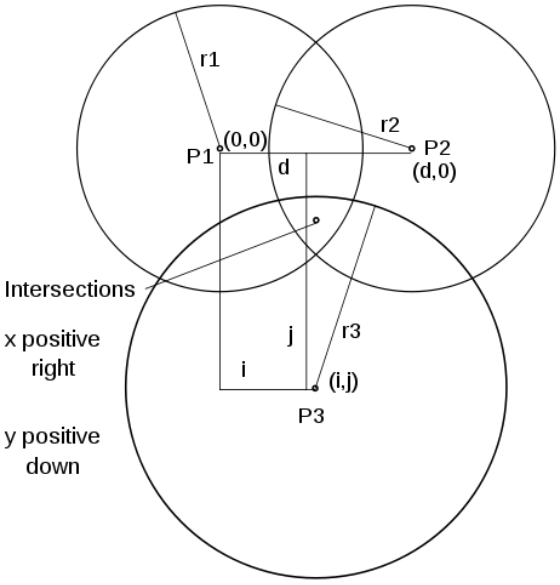

Figure 2: Using trilateration to predict the location of WiFi access-points (Wikipedia 2012*c*).

A few student teams demonstrated creativity by chosing to plot the apparent locations of access-points on a campus map, using user-defined layers in either *Google Maps* or *Google Earth*. While all on-campus access-points are located *inside* buildings or under overhanging balconies, a number of access-points appeared to be positioned in the middle of grassed areas, and in gardens. While physically unrealistic, this eventual placement is simply a result of the fact that WiFi signals are absorbed, reflected, and refracted as they pass through most building materials and vegetation. Even some atmospheric conditions, such as rain and high humidity, can have observable effects on the strength of arriving WiFi signals, and some teams' results would have been skewed if conditions on the days they sampled, were very different to those on the days they employed their predictions.

5 Student experiences

The goal of this project was to increase and assess students' understanding of the physical environment of wireless networks, transport layer protocols, and their ability to analyse and report on the observed effectiveness of their software. Although students were not formally surveyed, it is very clear from the many comments and questions raised on our unit's online forum, that the students enjoyed the nature of the project, and its use of mobile devices as, simply, mobile computers. They also particularly enjoyed devising approaches to a problem that appears innately solvable, but one that is so influenced by many external factors that only approximate solutions are possible.

With respect to the traditional networking requirements of the project, students were exposed to exactly the same details about the Berkeley sockets API, client-server connection, and communication patterns as they would have been had the project simply been undertaken only on desktop machines connecting to the wider Internet using wired infrastructure. However, what this project specifically introduced students to, was the need to develop robust software that did not crash in the presence of failure. Client-server software on wired infrastructure, even using wireless networks in laboratories or libraries *rarely fails*, and typically exhibits fast, regular, communication patterns – for the basic reason that its devices are rarely mobile.

This project not only required the students' code to make connections on-the-fly, but to detect frequent connection failures, Moreover, students had to design their software to ensure that queued communication was eventually transmitted. Students commented that they had never before had to write software that would experience failure, with I/O operations that could fail and require retrying. In our earlier units, such as *Operating Systems*, attention focuses on capturing and reporting the success or failure of read and write disk-based operations, but rarely (if ever) are students required to consider retrying failed operations, perhaps asynchronously. This is a consequence of *Operating Systems* being presented at (only) second-year level. There is a need for earlier units to more strenuously emphasize and assess the robustness of students' software and for units, such as *Computer Networks*, to set student projects in environments which do experience network failures.

Some of the other lessons learnt were quite surprising, and demonstrated the diversity of students' backgrounds – moreso than their ability to cope with certain problems. As stated in the previous section, the requirement that only the *apparent* location of access-points was required was foreign to some students, who insisted on knowing the *true* location of access-points (which we did not have nor seek). Students initially appeared to lack confidence in a being able to develop results in the project's first phase, and to then treat these as axiomatic for the second phase. Moreover, students learnt that the nature of wireless transmissions can be quite fickle, and that the often seen textbook explanations and simulations of signal propagation, in perfect circles, is never seen in practice (Kotz et al. 2004).

Students also had some difficulty using latitudes and longitudes as a Cartesian reference system, and mapping these to and from pixel-based coordinates on maps. We have no explanation for this difficulty, other than to report that most students had not undertaken an earlier unit in Computer Graphics, and that the problem sometimes arose because we are in the southern hemisphere, where latitudes increase (but become more negative) while moving southwards, or "down" the device's screen.

Students even discovered some other unique problems, totally by accident. One team found that their application kept crashing when walking through a certain area on campus, but worked well elsewhere. Extensive testing amongst the students revealed that a particular private WiFi access-point had used a full 32-byte SSID and was therefore not delivering the expected null-byte to signal the end of its SSID name. This sort of error might never have turned up in normal testing, and certainly never in simulation, but may have caused a similar application to crash when used in practice.

6 Summary

This paper has motivated and described in detail a project suitable for third or later year undergraduate students undertaking a unit in Computer Networks. The project requires students to design, implement, and analyse, a client-server software architecture, using a "hidden" server, desktop computers, and handheld, mobile, wireless devices. Students were required to employ an existing pre-defined protocol to communicate with a black-box server, and to design their own protocol communicating between their own two software components. Burd et al. (2012) have also summarised this work.

While this project employed (jailbroken) Apple iPod devices running iOS, as that was matched by our department's resources and knowledge at the time, there are no specific iOS requirements. Setting a similar project in coming years would likely benefit from the use of Android instead of iOS, and enable more students to use their own mobile devices. Interested readers are warmly invited to contact the author for copies of the software, and for instructions suitable to getting students started.

References

Andrus, J. & Nieh, J. (2012), Teaching operating systems using android, *in* 'Proceedings of the 43rd ACM technical symposium on Computer Science Education', SIGCSE '12, ACM, New York, NY, USA, pp. 613–618.
URL: *http://doi.acm.org/10.1145/2157136.2157312*

Bayzick, J., Askins, B., Kalafut, S. & Spear, M. (2013), Reading mobile games throughout the curriculum, *in* 'Proceeding of the 44th ACM technical symposium on Computer science education', SIGCSE '13, ACM, New York, NY, USA, pp. 209–214.
URL: *http://doi.acm.org/10.1145/2445196.2445264*

Burd, B., Barros, J. a. P., Johnson, C., Kurkovsky, S., Rosenbloom, A. & Tillman, N. (2012), Educating for mobile computing: addressing the new challenges, *in* 'Proceedings of the final reports on Innovation and technology in computer science education 2012 working groups', ITiCSE-WGR '12, ACM, New York, NY, USA, pp. 51–63.
URL: *http://doi.acm.org/10.1145/2426636.2426641*

Dabney, M. H., Dean, B. C. & Rogers, T. (2013), No sensor left behind: enriching computing education with mobile devices, *in* 'Proceeding of the 44th ACM technical symposium on Computer science education', SIGCSE '13, ACM, New York, NY, USA, pp. 627–632.
URL: *http://doi.acm.org/10.1145/2445196.2445378*

Guo, M., Bhattacharya, P., Yang, M., Qian, K. & Yang, L. (2013), Learning mobile security with android security labware, *in* 'Proceeding of the 44th ACM technical symposium on Computer science education', SIGCSE '13, ACM, New York, NY, USA, pp. 675–680.
URL: *http://doi.acm.org/10.1145/2445196.2445394*

Junco, J. & Mastrodicasa, J. (2007), 'Connecting to the net.generation : what higher education professionals need to know about today's students'.

Kotz, D., Newport, C., Gray, R. S., Liu, J., Yuan, Y. & Elliott, C. (2004), Experimental evaluation of wireless simulation assumptions, *in* 'Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems', MSWiM '04, ACM, New York, NY, USA, pp. 78–82.
URL: *http://doi.acm.org/10.1145/1023663.1023679*

McDonald, C. (1991), 'A network specification language and execution environment for undergraduate teaching', *SIGCSE Bull.* **23**(1), 25–34.
URL: *http://doi.acm.org/10.1145/107005.107012*

McDonald, C. (2009), Linking simulation and practice, *in* 'CSEDU 2009 - Proceedings of the First International Conference on Computer Supported Education', Vol. 2, INSTICC Press, New York, NY, USA, pp. 258–263.

Oblinger, D. & Oblinger, J. L., eds (2005), *Educating the net generation*, EDUCAUSE, Boulder, CO.

Riley, D. (2012), Using mobile phone programming to teach java and advanced programming to computer scientists, *in* 'Proceedings of the 43rd ACM technical symposium on Computer Science Education', SIGCSE '12, ACM, New York, NY, USA, pp. 541–546.
URL: *http://doi.acm.org/10.1145/2157136.2157292*

Wikipedia (2012*a*), 'Geolocation', http://en.wikipedia.org/wiki/Geolocation.

Wikipedia (2012*b*), 'Triangulation', http://en.wikipedia.org/wiki/Triangulation.

Wikipedia (2012*c*), 'Trilateration', http://en.wikipedia.org/wiki/Trilateration.