The 16th Winona Computer Science Undergraduate Research Symposium

April 27, 2016

9:30am to 12:30pm

Kryzsko Commons Purple Rooms 247 & 248

Winona State University
Winona, MN

Sponsored by the Department of Computer Science at
Winona State University

WINONA
STATE UNIVERSITY
Computer Science Department
http://cs.winona.edu

Table of Contents

Effectiveness of Children Online Privacy Strategies

Komal Bansal
Winona State University
April 22, 2016
kbansal14@winona.edu

ABSTRACT

The Children's Online Privacy Protection Act (COPPA) is a Federal Trade Commission (FTC) approved regulation directed towards websites/apps that collect and use personal information (PI) of children under 13 years of age. Under COPPA, FTC has approved a fees-based Safe Harbor program, in which websites/apps would be subjected to the disciplinary procedures of the Safe Harbor in lieu of the FTC enforcement. This study conducted interviews, surveys, and a scenario-based usability study with parents of children under 13. The study investigated the effectiveness and awareness of COPPA's regulations and COPPA's Safe Harbor program. COPPA requires websites/apps to be completely transparent with their activities while engaging and collecting PI from children under 13 years of age, and to let parents have full control over their children's PI. However, this study found out that parents are neither aware of COPPA and Safe Harbors nor do parents know if websites/apps are following COPPA's mandatory guidelines. Due to lack of such awareness, parents continue to feel insecure about their child's online privacy, and COPPA's regulations remain less effective among parents of children under 13.

General Terms

Reliability, Security, Human Factors, Privacy

Keywords

COPPA, FTC, Children Online Privacy Protection Act, Safe Harbors, PRIVO, TRUSTe, kidSAFE, iKeepSafe, Aristotle, ESRB, CARU

1. INTRODUCTION

The Children's Online Privacy Protection Act (COPPA) is a United States Federal Act that applies to websites/apps that collects any kind of personal information (PI) from children under 13 years of age. PI includes first name, last name, physical address, online contact information, telephone number, social security number, a child's photograph, audio file, video file, geolocation information sufficient to identify the location of the child, and user name sufficient to identify the child. According to COPPA, websites/apps that collect any kind of PI need to follow these guidelines [1]:

1. Provide a clear and complete privacy policy on their website
2. Obtain verifiable parental consent before collecting private information about their children
3. Provide parents access to the information collected on their children and allow them to withdraw permission on future collection on their children's information
4. Maintain confidentiality of the information collected
5. Minimize the retention period of children information for as long as is necessary, and delete the data responsibly.

From 2008 through 2015, the Federal Trade Commission (FTC) fined several companies for violating COPPA. As illustrated in Table 1, some of the companies like Sony, Disney's Playdom, Yelp, and TinyCo were fined due to collecting PI from children without verifiable parental consent. Echometrix failed to adequately provide parents on how they sold the information collected from their children to third party. RockYou was fined due to collecting PI from children without verifiable parental consent. Retro Dreamer and LAI were fined because they allowed third-party ad networks to collect persistent identifiers in order to serve targeted ads on the child-directed apps. The FTC has proven to be strict with websites/apps that violated COPPA regulations by imposing steep fines and punishments.

Table1: Companies fined by the FTC for violating COPPA

Year	Company	Fine Amount
2008	Sony BMG Music Entertainment	$1,000,000
2010	Echometrix	$100,000
2011	Disney's Playdom	$3,000,000
2012	RockYou	$250,000
2014	Yelp	$450,000
2014	TinyCo	$300,000
2015	Retro Dreamer	$300,000
2015	LAI	$60,000

Hypothesis 1: Awareness

The primary goal of COPPA is to give parents full control over their children's online activity. Thus COPPA imposes strict guidelines on websites/apps to provide transparent detailed

Proceedings of the 16th Winona Computer Science Undergraduate Research Seminar, April 27, 2016, Winona, MN, US.

privacy policies, and obtain verifiable parental consent before collecting any PI from a child under 13 years of age. However, the majority of parents are not aware of any law that protects their child's online privacy. Thus, the first hypothesis of this research is as follows: *Parents continue to feel insecure about their child's online safety, and they will assume that any website can be harmful to their child, even when in reality, that website might be COPPA compliant.*

COPPA has also provided "Safe Harbor" provisions to encourage websites/apps to follow COPPA without being formally investigated by the FTC and law enforcement. The following section explains the role of Safe Harbors and their relevance to verifiable parental consent.

1.1 Safe Harbor provision

Under COPPA, websites/apps are subjected to the review and disciplinary procedures provided in the Safe Harbor's guidelines in lieu of the formal FTC investigation and law enforcement. As of April 2016, the FTC has approved seven Safe Harbors: ESRB, PRIVO, CARU, Aristotle, kidSAFE, iKeepSafe, and TRUSTe. Generally, Safe Harbor certified websites contain their membership "seals" on their websites, which provide assurance to the parents that these websites/apps are COPPA compliant. For instance, the famous Pokemon website [4] has the seal of FTC approved Safe Harbor, ESRB, shown in Table 2, certifying that Pokemon Company International, Inc. is COPPA compliant.

There are seven different FTC approved Safe Harbors, each with their own seal, as shown in Table 2. The purpose of a Safe Harbor seal is to assure parents the following [1]:

- The website has posted a privacy policy and the privacy policy describes in detail how the information collected from the child is used and shared.
- The website is reviewed periodically along with unannounced monitoring reviews by the Safe Harbor.
- The website provides direct notice to parent before collecting their child's PI and then obtaining verifiable parental consent.
- The website gives parents the choice of consenting to the operator's collection and internal use of a child's information, but prohibits the operator from disclosing that information to a third party.
- The website provides parents access to their child's PI to review and/or have the information deleted.
- The website minimizes the retention period for children's PI as long as is necessary, and deletes the data responsibly.

Hypothesis 2: Judge a website

Phishing is an attempt to acquire sensitive information from an online user by masquerading as an original website's page and asking users to enter the information, which is then emailed to the hacker. About two million users gave information to spoofed websites/apps resulting in direct losses of $1.2 billion for U.S. banks and card issuers in 2003 [8]. A user is generally deceived by a phishing website due to lack of knowledge, visual deception, and lack of attention [9]. One of COPPA's main objectives is to provide parents with detailed information about how and what PI websites/apps collect and use from children under 13. These websites/apps are also required to have a detailed privacy policy describing how the websites/apps gathers, uses and shares the PI of a child, keeps the PI confidential and keeps the retention period of the PI to the minimum. Moreover, if a website is under the Safe Harbor program, then that website should not only have detailed privacy policy, but may also have a Safe Harbor seal, generally at the homepage or at the privacy policy page. Such detailed policy and/or a seal provides a quick assurity to the parents that the website is COPPA complaint and under the constant monitoring/auditing of the Safe Harbor. Yet, many parents are neither aware of any Safe Harbor nor are parents aware of any Safe Harbors' seals. The second hypothesis is the following: *Parents continue to judge the safety of a website based on factors other than a Safe Harbor Seal or even reading the website's privacy policy.*

Table 2: Safe Harbors and their Seals

No.	Safe Harbor	Seal
1	ESRB	PRIVACY CERTIFIED Kids ™ ESRB
2	PRIVO	PRIVO® PRIVACY CERTIFIED
3	CARU	BBB CARU Kid's Privacy Safe Harbor
4	Aristotle	INTEGRITY™ KIDS PRIVACY COMPLIANCE CERTIFIED
5	kidSAFE	kidSAFE+ COPPA CERTIFIED ™
6	iKeepSafe	iKEEPSAFE PRIVACY COPPA SAFE HARBOR
7	TRUSTe	TRUSTe KIDS PRIVACY

1.2 Verifiable parental consent

COPPA requires websites/apps to notify and obtain verifiable parental consent before collecting any PI from children under 13 years of age. Parents can provide their *verifiable* parental consent through any of the following process [1]:

1. Signing a consent form and send it back to the website operator via fax, mail, or electronic scan.

2. Use a credit card, debit card, or other online payment system that provides notification of each separate transaction to the account holder.
3. Call a toll free number staffed by trained personnel.
4. Connect to trained personnel via a videoconference.
5. Provide a copy of a government issued ID and delete the identification from the records once the verification process is complete.

Hypothesis 3: Response to Verifiable parental consent email

Protecting a child's PI is utmost important because the consequence of a child's identity theft can be devastating. Online predators lie with children, make friends with children, and ask the child's PI from them. Online predators can misuse different kinds of PIs like first name, last name, email address, home address or social security number in many ways, like, creating a fake profile, full credit history, financial history, IRS history, public record, and even criminal record, all before the child even applies for her/his own government ID [7]. COPPA requires websites/apps, which collect PI from children under 13, to obtain verifiable parental consent before obtaining any PI from the child. This can be done by websites either independently or through Safe Harbors program. Each of the Safe Harbors have unique mechanisms to obtain verifiable parental consent on behalf of websites/apps. Several Safe Harbor compliant websites/apps initiate a verifiable parental consent process by asking the child's name, date of birth, and parent's email address. An email is then sent to the parent seeking verifiable parental consent and explaining why they are being asked to provide verifiable parental consent. Finally, the third hypothesis is as follows: *However, most parents will not respond to such an unsolicited email and ignore it considering it a spam.*

2. Background Research

Children are most exposed to the dangers of the Internet and least able to protect themselves [11] but commercial websites/apps have been found to be more attractive to younger consumers than older consumers [12]. According to the "Deleting the Predators Act of 2006" [12], schools and libraries are required to restrict access of "commercial social networking websites" and "chartrooms" to minors. Some research has been done to analyze threats to children from online dangers but no research has been done to understand, from parents' perspectives, the effectiveness and awareness about COPPA.

COPPA was passed by the U.S. Congress in 1998 and took effect in April 2000. It is under the Federal Trade Commission (FTC). On December 19, 2012, the FTC issued its final rule amendments concluding its review of COPPA [1]. The revised rules were in light of the changes in the online technology regulations. The final COPPA amended rule included modifications to the definitions of an operator, PI, and website or online service directed to children. The amended rule also updated the requirements set forth in the notice, verifiable parental consent, confidentiality and security, and Safe Harbor provisions, and added a new provision addressing data retention and deletion. Since the original COPPA was revised, little research has been done on the effectiveness of COPPA, Safe Harbors, verifiable parental consent measures, and safety of websites/apps directed towards children under 13 years of age.

3. Methodology

3.1 Usability Study

Usability Studies have been increasingly used since the 1990s to evaluate the effectiveness of user interfaces [5][6] and to obtain feedback from users to improve the existing user interfaces. This research implemented a usability study to test the effectiveness and awareness of a set of COPPA and Safe Harbors strategies. The research was conducted with parents of children under 13 using questionnaires, interviews, and empirical methodology (usability assessed by testing the interface with real users) to evaluate:

- Hypothesis 1: Awareness
- Hypothesis 2: Judging a website
- Hypothesis 3: Response to verifiable parental consent email.

The following sections will describe briefly the participants recruitment process and the methodology used for testing the above stated hypotheses.

3.2 Participants

Ten participants were recruited from two U.S. states, Minnesota and Wisconsin. A letter [Appendix 1] was sent through email to recruit the eligible parents explaining briefly about the purpose and scope of the study. The eligibility of a parent was based on the following three factors:

- Their child should be under 13
- Their child can use an app or a website
- The parent is concerned about their child's privacy

The usability study was performed as per the participant's convenient place and the study was conducted using a laptop or a smartphone. Prior to the study, the participant signed a consent form. The consent form stated that participation was voluntary and the study was conducted to judge the usability of websites/apps and Federal regulations. The study was approximately fifteen minutes long. The participants were presented with a questionnaire and scenarios. The questionnaire and scenarios will be briefly described in the Section 3.3. 70% of the participants were female. The age of participants ranged from 18 through 54 years. The age of their children varied from 3 to 12 years.

3.3 Usability Goals

3.3.1 Goal 1

The goal to support Hypothesis 1 was defined to be *90% of the parents are not aware of any law that protects their child's online activities*. This goal was tested by interviewing and asking questions to parents who have children under 13. The questions are mentioned in Table 3 below.

Table 3: Questions for Goal 1

1	Which of the following are you familiar with? The options included COPPA, names of all seven Safe Harbors, COPPA's seal program, and definition of COPPA.
2	"Websites/Apps, which are collecting PI of children under 13 years of age, should have special protection rules for those children." How would you rate your support or opposition to this statement?
3	Do you think that websites, that collect personal information (PI), should have an age requirement to create an online account?

3.3.2 Goal 2

The goal to support Hypothesis 2 was defined to be *90% of parents judge the reliability of a website by something other than a Safe Harbor seal or a privacy policy.* Parents were shown a COPPA compliant website and then asked the questions mentioned in Table 4.

Table 4: Questions for Goal 2

1	Are you familiar with this website before?
2	Do you think your child is familiar with this website?
3	Do you trust this website? Please explain why or why not do you trust this website?

3.3.3 Goal 3

The goal to support hypothesis 3 was defined to be *80% of the parents would not respond to an unsolicited email seeking parental consent. They would consider it spam and ignore it.* In this study, the instructor sent an email to the parent from an app seeking verifiable parental consent. During the interview and questionnaire session, the parent was asked the follow up questions mentioned in Table 5.

Table 5: Questions for Goal 3

1	Would you open this email?
2	When you receive an email from a website or app asking for your parental consent, what will you think and do?
3	How comfortable are you when your child shares your email address with online websites/apps?

4. Results and Analysis

4.1.1 Hypothesis 1

Parents were asked questions, as shown in Table 3, about their familiarity with COPPA or any Safe Harbors. Following were the results from the interviews and questionnaire:

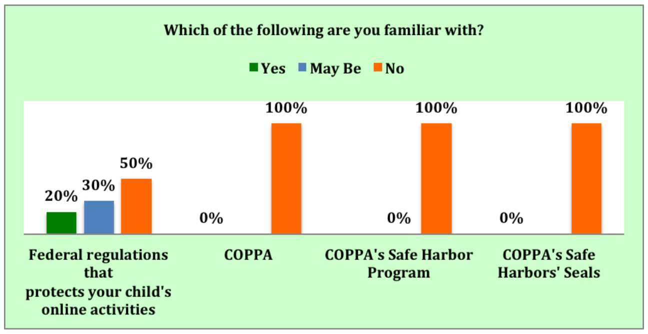


Figure 1: Familiarity with COPPA

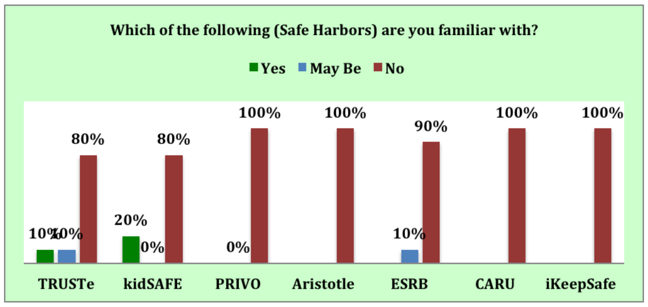


Figure 2: Familiarity with Safe Harbors

According to Figure 1,

- Only 20% of the parents said that they were familiar with an existing US Federal regulation that protects children under 13 (because they were *educated* by Girl's Scouts program). However, none of the parents knew about COPPA.
- 0% of the parents knew about FTC approved Safe Harbor program, or Safe Harbors' seals.

According to Figure 2,

- 0% of the parents knew about PRIVO, Aristotle, CARU, and iKeepSafe Safe Harbors.
- Only 10% of the parents knew about ESRB.
- 20% of the participants knew about TRUSTe and kidsSAFE Safe Harbors.
- However, parents who knew about TRUSTe and ESRB, knew them because of their business other their than COPPA Safe Harbors.

4.1.2 Hypothesis 2

The instructor showed a COPPA compliant Safe Harbor-sealed website to the parent, and asked why or why not would they trust the website, as shown in Table 4.

According to Figure 3:

- 80% of the parents said they trusted the website by looking at the content of the website. Only 20% of the parents trusted the website by looking at the privacy policy.

- 70% of the parents considered media logos like:

as seals/certificates to trust the website but none of the parents looked for a Safe Harbor Seal.

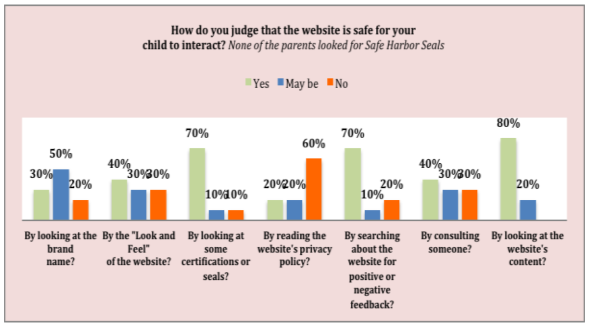


Figure 3: How parents judge a website?

4.1.3 *Hypothesis 3*

The participant was sent an email, seeking their verifiable parental consent. When the parent received the email, the instructor asked questions mentioned in Table 5. Following were the observations:

- 60% of the parents said they would not at all ignore the email, which asks for verifiable parental consent, only because their child's name was in the email. Otherwise, parents will ignore the email. This result contradicted this study's hypothesis, which stated that most of the parents will completely ignore emails seeking parental consent.
- However, as shown in Figure 4, 50% of the parents said they would be somewhat or very uncomfortable if their child shared their email address with websites/apps.

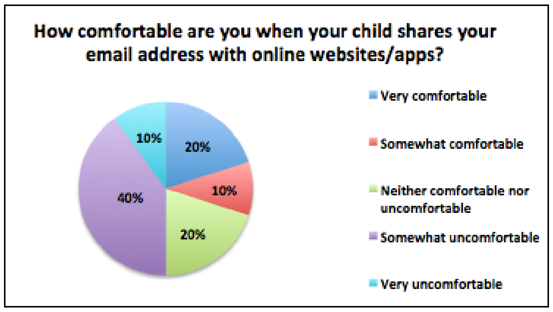


Figure 4: Are Parents comfortable if their child shares their email address online

4.1.4 *Other observations*

During the usability study, few other interesting observations about *parents' perspective towards their children's online privacy* were also recorded:

- *Age requirement for online websites*
  As shown in Figure 5, 90% of the parents think that websites/apps that collect PI should have an age requirement to create an online account but they are not aware that COPPA is mandating this regulation on the websites/apps. Moreover, as shown in Figure 6, 90% parents very strongly or somewhat strongly support that websites/apps that collect PI from children under 13 should have special protection rules for those children but parents are not aware that COPPA is already mandating the websites/apps to obtain verifiable parental consent from parents whenever websites/apps collect PI from children.

- *Children's personal information*
  Only 40% of parents considered *first name* and *audio (containing child's voice)* as PI. COPPA strictly considers both as a child's PI and websites/apps that collect first name and audio are required to obtain verifiable parental consent before collecting or storing the PI at their server.

- *Control over children's online privacy*
  According to Figure 7, 80% of the parents said that they would like to have their child have *some* control over their online privacy. Only 30% of the parents said that they do not want the government to have *full* control over their child's online privacy. However, COPPA's primary goal is to provide *only* parents with full control of their child's online privacy.

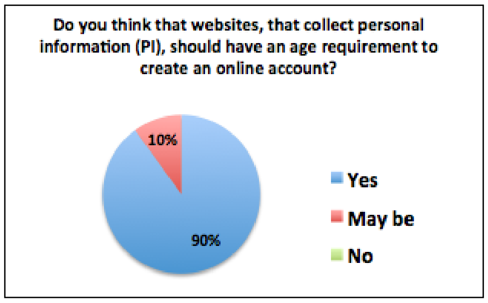


Figure 5: Age Requirement to create online account

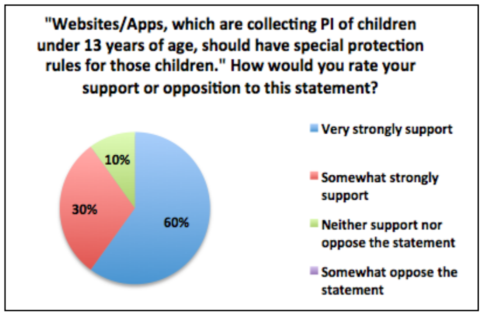


Figure 6 Special protection rules online for children under 13

- *Trust towards their children's online activities*
  In one of the scenarios, parents were shown a COPPA Safe Harbor certified website which *they had never seen before*. When parents were asked if their child had seen that website before, as stated in Table 4, 70% of the parents said *No, their child has also never seen that website*. The age of the children of these parents ranges from 5 to 12 years. This result seemed overly positive. An extended usability study could be performed with children to validate the same results.

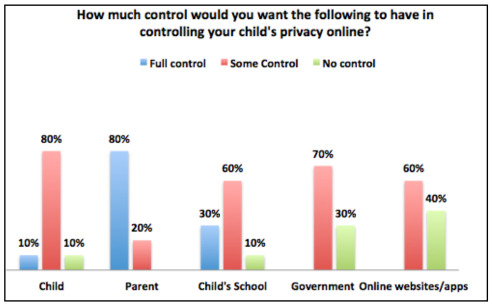


Figure 7 How much control of Child's PI

5. CONCLUSION

COPPA's primary goal is to provide parents full control of their child's (under 13) online interaction. COPPA has good intentions to protect children but parents do not know about these intentions and continue to feel insecure about their child's online privacy. That is why educating parents about children online privacy strategies like COPPA, verifiable parental consent, and FTC approved Safe Harbors is essential for these strategies to be effective.

Parents do not know about COPPA and parents also do not know why they receive emails or other forms of communications seeking parental consent. In this research, when a parent received an email seeking verifiable parental consent, the parent did not know that the email was sent due to COPPA's regulation. Most of the parents would not ignore emails seeking parental consent because their child's name was in the email subject. However, this format of having the child's name in the email subject might not be followed by all websites/apps and hence, parents would ignore such emails. In any case, parents should be made aware of why they receive such emails or any other form of communication from websites/apps to obtain verifiable parental consent. Such education and awareness among parents would help websites to obtain parental consent faster, giving parents full control over their child's data.

Apart from providing full control to parents over their child's data, COPPA also provides an effective way to judge the reliability of a website by mandating websites/apps to have a descriptive and easy to understand privacy policy and/or websites to optionally have Safe Harbor seals if the website is under the Safe Harbor program. However, not all websites/apps are under the Safe Harbor program nor do all websites/apps have Safe Harbor seals. But those websites/apps that do have seals on their homepage could provide assurity to parents about their legitimacy and safety. Moreover, parents usually may not read the privacy policy but if they are aware that the websites/apps have a privacy policy link on their home page and on all the pages that collect PI from the child, parents could better judge the reliability of that website. The parent may open the policy page and skim the policy to have an idea that the policy is discussing COPPA, children under 13, verifiable parental consent, child's data confidentiality, and minimum retention period. Making sure that the privacy policy exists and skimming the policy would make it easier for parents to feel safe about a website.

COPPA is somewhat complicated to follow. That is why the FTC has provided Safe Harbor support to websites/apps to implement COPPA under Safe Harbor monitoring without worrying about any litigation or fines. Just as the FTC has provided Safe Harbor support to websites, it may be advantageous for the FTC to ask schools to mandate compulsory COPPA education to parents during conferences or parent-teacher sessions. Having a compulsory curriculum would ensure that all parents are educated enough about COPPA and supported verifiable parental consent processes and could therefore easily judge the reliability of a website.

6. FUTURE WORK

This study discussed the need and importance of educating parents about the Children Online Privacy Act (COPPA) and FTC approved Safe Harbors. However this study did not evaluate the reliability of Safe Harbors and their auditing and monitoring process. Also, COPPA does not require websites/apps to investigate the age of the visitors. Thus, COPPA doesn't apply when kids lie about their age or find other workarounds, and parents need to understand that their children's data is not being monitored under COPPA's regulation in these cases. Companies like Facebook, Instagram, Snapchat, and Twitter have age restrictions for children under 13 possibly because COPPA may be complicated for them to implement and obtaining verifiable parental consent may not be feasible at such a large scale. However, several kids under 13 are on such websites, violating the age policy by lying with or without their parents' help. A lie is a lie, be it said online or offline. Since children do not see much difference between the digital world and the real world, letting them think "lying online is okay," might confuse them in the real world. Maybe COPPA and such impactful companies like Facebook, Twitter, Snapchat, and Instagram could work together one day in a way that parents and children do not find the need to lie online.

This study focused on parents of children under 13 but I strongly feel that parents of children above 12 years and below 18 years of age are facing much more challenges than the rest.

App stores like Google Play store and Apple store have strict guidelines to verify the legitimacy and safety of an app that is directed towards children under 13. However, in Apple store, a child directed app has to comply with two components: COPPA and Parental gate. Many developers find it complicated to comply with both regulations. Research could be done to understand the difference between COPPA and Parental gate regulation and why Apple cannot combine both regulations, making it easier for a developer to get approval for their app.

In December 2015, Mattel Inc. was charged by a class action lawsuit [10] alleging that its "Hello Barbie" records children's conversation (including children who do not own the Hello Barbie, but are still around some other child's Hello Barbie) without verifiable parental consent, directly violating COPPA. Parents were concerned about scenarios where hackers could obtain PI from the child through the Hello Barbie, storing the conversation on the hacker's server instead of Mattel's own server and misusing the data to their own advantage. This is a scary scenario where COPPA meets the Internet of things. These toys are using technology to reach children's private life without providing proper protection to them or their family's privacy. Unless there are proper measures to protect the privacy and security of the users, maybe such toys should be banned, at least for the most vulnerable users, that is, children under 13 years of age.

7. MOTIVATION OF RESEARCH

The inspiration behind this research work is my childhood. When my sister, *Kanika Jain*, and I were given a computer with Internet access eleven years ago for the first time, my mother, *Reeta Jain* and my father, *Naveen Jain*, used to constantly worry about our interaction on Yahoo messenger and Orkut. At that time, Kanika was 12 years old and I was 15 years old. Our parents wanted to monitor our online interaction with strangers but had no way to do it. They used to say that they did trust us but they did not trust the strangers we interacted with, on the Internet. They were not computer savvy parents eleven years ago, but they were aware of the concept of online predators in their own way. I wish, at that time, my parents could have had measures to judge websites/apps and could have had the ability to provide verifiable parental consent to websites/apps that asked for children's PI. However, now, when parents do have measures to judge the reliability of websites, when parents can have access to their child's data shared with websites/apps and when there are strict Federal regulations to protect children's online privacy, parents are still unaware and insecure about their child's online privacy.

8. ACKNOWLEDGMENTS

The title may include only one author but there are several people whose contribution immensely helped in this research. Thanks to *Dr. Joan Francioni* and *Dr. Gerald Cichanowski* for teaching Human Computer Interaction and Security and Privacy course, respectively. Dr. Joan spent an immense amount of time guiding me how to pick a research topic, how to narrow it down, and how to make it practically doable. Thanks to *Dr. Iyengar*, my program advisor, for including me in his Middle School Computer Programming course and allowing me to teach 7th and 8th graders. Teaching kids helped me develop insights into how children and parents think in terms of online privacy. Thanks to Dr. Iyengar and *Dr. Francis Mann* (Chemistry professor) for helping me recruit parents for the research. Thanks to *Rahul Bansal*, my husband: Through his critical thinking and Socratic approach towards problem solving, he helped me from my predisposition of trying to solve problems that were not supported by my methodology. Thanks to Grace Hopper Conference for connecting me to *Kaliya Young*. Kaliya is currently working as the privacy and identity research Program Manager at the US Department of Homeland Security Science and Technology. Kaliya introduced me to Denise Tayloe, PRIVO's founder and CEO. Even for me, the concept of COPPA and Safe Harbors was complicated to understand in the beginning. Denise Tayloe and her team, *Shauna Leff*, *Carol Altarescu*, and *Claire Quinn* proactively provided their answers and feedbacks to my research questions. Thanks to *Shai Samet*, founder and President of kidSAFE Seal Program, for also providing his feedback and suggestions to my research work.

9. REFERENCES

[1] COPPA, Children's Online Privacy Protection Act 1998, Revised 2013.

[2] Federal State Commission: 2008- 2015. *Children Online Privacy Protection Act*

[3] Liccardi, Ilaria, et al. "*Can apps play by the COPPA Rules?*." *Privacy, Security and Trust (PST)*, 2014 Twelfth Annual International Conference on. IEEE, 2014Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.

[4] Pokemon Company International Inc., 2016. http://www.pokemon.com/us/

[5] Usability, 2016. www.Usability.gov.

[6] Nielsen, J., and Mack, R. L. (Eds.) (1994). Usability Inspection Methods, John Wiley & Sons, New York.

[7] iKeepSafe, 2016, Why protecting your childs identity is important, http://ikeepsafe.org/products/idefend/why-protecting-your-childs-identity-is-important/

[8] Litan, A. Phishing Attack Victims Likely Targets for Identity Theft. Gartner Research (2004).

[9] Dhamija, Rachna, J. Doug Tygar, and Marti Hearst. "Why phishing works." *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006.

[10] Courthouse News Service, 2015. *Talking Barbie Invades Privacy*

[11] Montgomery, Kathryn C. (2000), "Youth and Digital Media: A Policy Research Agenda," Journal of Adolescent Health, 27 61–8.

[12] Library of Congress (2007). "Deleting Online Predators Act of 2006," (accessed February 19, 2007) [available at http://thomas.loc.gov/cgi- bin/query/z?c109:H.R.5319].

Comparing Linux Operating Systems for the Raspberry Pi 2

John Bellows
Department of Computer Science
Winona State University
Winona, MN 55987
(612) 220-1321
JBellows13@winona.edu

ABSTRACT

The Raspberry Pi 2 is a single-board, low-cost computer capable of running a GNU/Linux desktop environment on a low-power ARM processor. Because the RPi has limited performance hardware due to its size and cost, it is important to use an operating system that best utilizes the systems' available resources. There are three categories of tasks every operating system must manage to ensure correct behavior of the system: memory management, secondary storage management, and process management. Four popular Linux distributions were chosen for comparison using open-source benchmarking software, they include: Arch, Debian, Fedora, and Ubuntu Linux. Results showed significantly better memory access times for Arch and Ubuntu over Debian and Fedora. In the multi-thread DBench test, Ubuntu produced superior disk access times. The processor results were inconclusive as each operating system performed similarly. No one operating system clearly outperformed the others in all areas, leading to the rejection of the hypothesis.

Keywords

Benchmark; Raspberry Pi; Linux; Operating Systems.

1. INTRODUCTION

In the three years since the first version [9] of the credit-card sized Raspberry Pi (RPi) single-board computer was released the platform has grown incredibly popular, sold millions of units, and inspired a range of similar small form factor single-board computers. A driving force behind the popularity of the RPi is its low cost ($35) and use of open-source software, including operating systems. In traditional computing the choice of operating system, especially in the case of open-source, can largely be a matter of personal preference. However, given the limited hardware capabilities of the RPi the efficient use of resources by the operating system is particularly important.

The main functions of an operating system are to manage hardware, run applications, and provide an interface to the user [14]. The second generation of the RPi, the Raspberry Pi 2 Model B, features an energy-efficient Cortex A7 multi-core processor, one gigabyte of RAM, and utilizes the ARMv7 architecture [11]. This enables a wider range of GNU/Linux operating systems to run on the current RPi's architecture compared to the previous single-core, 512 gigabyte RAM based generation [13]. Arch, Debian, Fedora, and

Ubuntu [12] [1] [4] [17] are a few of the major Linux distributions to offer ARMv7 versions of their operating system. Additionally Unix-like distributions such as FreeBSD [5], and a version of Windows 10 called Windows 10 IoT Core [2] have been developed for use on the RPi. This increase in available operating system choices raises a question: how does the choice of operating system affect performance on the RPi?

There are several ways in which an operating system can be organized, however there are three categories of tasks every operating system must manage to ensure correct behavior of the system: memory management, secondary storage management, and process management [14]. Memory management handles the reserving and freeing of space in main memory for processes. Secondary storage management is concerned with the file system as well as preserving data in mass storage devices. Process management deals with adding and deleting processes (jobs) and handling inter-process communication. How a particular operating system implements each of these categories can have an effect on system performance.

In order to compare the performance between operation systems benchmarks need to be taken. Benchmarking provide a way to obtain a metric that can then be used to access how well a system performs [4]. To this end a benchmark must be straightforward and run on all systems tested in order to provide meaningful results. Additionally the test should be reproducible and simulate real-world usage of the system. Open source benchmarking tools are ideal for this purpose as they are cross-platform, maintained by a large community, and are freely available. Operating system performance will be measured in the following areas: memory access, the ability to handle allocation and deallocation requests to main memory; disk access, how quickly information is retrieved and stored from the disk; and processor time, the measure of CPU performance under heavy computation [3].

Limited research has been conducted on the performance of the different operating systems available for the RPi. Most work focuses on either comparing the performance of different single-board computers with the RPi [8], or is anecdotal in nature. However, some work has been done to compare Raspbian (the operating system that ships with the RPi) with Pidora (an optimized version of Fedora Linux) [3]. The goal of this paper is to compare the performance of several Linux based operating systems using an empirical approach. Benchmarks will be performed on Arch, Debian, Fedora, and Ubuntu Linux distributions using the Raspberry Pi.

Hypothesis-Arch Linux ARM has faster memory requests, disk access, and processing time than other Linux operating systems.

2. BACKGROUND RESEARCH

Most research comparing the performance of single-board computers focuses on testing the hardware itself. The goal of this paper is to compare the performance of software, in the form of

Proceedings of the 16th Winona Computer Science Undergraduate Research Seminar, April 27, 2016, Winona, MN, US.

operating systems, on the same piece of hardware (the RPi). An operating system can be organized in a variety of ways that influence overall system performance making it important to measure the effect a given operating system has on the use of system resources.

In a similar work, “Performance Comparisons of Operating Systems for the Raspberr Pi”, Joshua Dinndorf examines two operating systems: Raspbian and Pidora. These operating systems were chosen due to their optimization specifically for use on the RPi [3]. The research in this paper differentiates itself from that of previous work in that all operating systems tested are non-optimized ports of major distributions and therefore more closely resemble conventional Linux environments.

3. METHODOLOGY

A research project is conducted to test the hypothesis. This project tests three functions of operating system: process management, memory management, and secondary storage management with open source synthetic benchmark tools. As it is difficult to completely separate the influence of one area of operating system function from the others, however benchmarking tools were chosen with as much isolation as possible in mind.

3.1 Hardware and Software

The hardware being tested is a Raspberry Pi model 2 B+, it features a 32-bit ARM Cortex-A7 processor and 1GB RAM [12]. The disk used to install each operating system is a 16 GB class 10 micro SD card. Class 10 SD cards [15] offer the highest transmission speeds in a card currently compatible with the Raspberry Pi. Using this card will minimize any performance bottlenecks caused by the OS installation media.

Table 1. Hardware specs for the Raspberry Pi 2

Hardware	Raspberry Pi 2 B+
CPU	Cortex A7
Architecture	ARMv71
Cores	4
Clock Speed	900 MHz
GPU	Videocore IV
Memory	1GB
Secondary Disk	16GB Class 10 SD card

Table 2. System information of tested operating systems

	Arch	Debian	Fedora	Ubuntu
OS Version	Arch ARM	Debian 8	Fedora 23	Ubuntu 14.04
Kernel	4.1.19-5 ARCH	3.18.0 trunk-rpi2	3.18.0-20 rpi2	4.1.20-v7+
Compiler	GCC 5.30	GCC 4.92	GCC 4.8	GCC 5.3.1
Processor Speed	0.90GHz	0.80GHz	0.90GHz	0.90GHz
Package Manager	Pacman	Apt-get	Dnf	Apt-get
Memory	922MB	925MB	925MB	923MB

The Phoronix Test Suite (PTS) [11], a cross-platform suite of over 200 test profiles, will be used to perform benchmarks. PTS was chosen because of its GNU GPLv3 license, ability to run on any system containing PHP CLI (php5-cli, php5-gd, and php5-common packages) and a GCC compiler, and the support PTS offers to install any additional dependencies using the operating system’s native package management system [11].

All operating systems tested were configured in a minimal server-style setup without a desktop environment installed. A desktop environment is a bundle of components that act as the graphical user interface for an operating system by providing a windowing system and pointer among other features. The exact desktop environment installed on a Linux system often differs between distributions and may use system resources more or less efficiently depending on the desktop environment installed. The decision to run the tests from the command shell instead of a GUI was made in order to limit any unnecessary use of system resources that could negatively impact the test results.

3.2 Memory

Main memory management is a vital function performed by an operating system. For a program to be executed it must be mapped to addresses and loaded into memory [14]. There are many different memory management schemes used in computing systems. These schemes must keep track of the memory that is currently being used, what processes need to be loaded into memory, and which blocks are designated free space. How an operating system implements memory management influences the read and write speeds to main memory.

RAM-Speed is a benchmark that measures the performance of a systems RAM by allocating varying amounts of memory space, and reading or writing to it in blocks [7]. RAM-Speed operates in two modes, integer and floating point and computes block usages with either copy, scale, add, triad, or average schemes. All schemes are run for both integer and floating-point mode to get as full a picture of ram performance as possible.

Another benchmark used to measure memory performance is STREAM. STREAM is a benchmark designed to measure the bandwidth of a hierarchic memory subsystem for read, write, and read-modify-write access. Each mode generates a memory bandwidth curve of varying vector lengths in a compiler-optimized loop. The STREAM benchmark is chosen to give a good indication of the cache bandwidth performance of operating systems on the RPi.

3.3 Secondary Storage

Secondary storage systems make up the majority of memory in a computer system and are responsible for the efficient accessing of the disk to locate, read, and write data [14]. The difficulty when benchmarking a file system located on a secondary storage device is to isolate it as much as possible from main memory and CPU intensive tasks. While benchmarking a file system we are interested in the bandwidth and latency when reading from and writing to the disks in various sized blocks.

The first benchmark used is Dbench, an emulation of a server in which a large amount of files with varying sizes have to be created, written, read, and deleted [16]. When run, Dbench creates a number of parallel clients as specified by input parameter. The result of the test is an average latency of the operations executed by each process in megabytes per second.

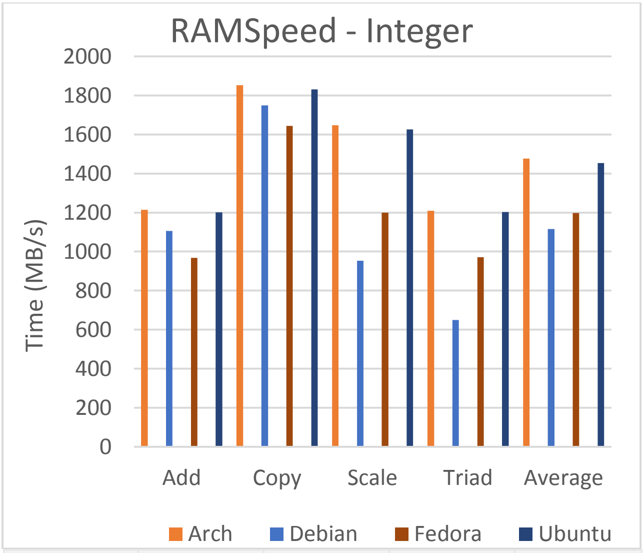
Lastly the IOzone benchmarking tool was chosen because of its ability to perform sequential and random I/O. IOzone is a file system benchmark tool that reads and writes to a file while running multiple instances in parallel [10]. A broad indication of file system performance is obtained from using IOzone to measure both sequential and random on-disk read and write latencies.

3.4 Processor

The processor is an important part of a computer system, without a processor to execute instructions a program cannot run [14]. Because an operating system manages processes and the resources they use, testing the processor's ability to execute computationally complex processes is a good indicator of the effect an operating system has on the overall system's performance.

The first benchmark used to test the processor is PHP-Bench, a testing suite for the PHP interpreter [3]. PHP-Bench performs a large number of simple tests while recording the time taken to complete them. Since PHP is widely available, the ratio between number of tests executed and the time taken to complete them is a good indicator of processor performance under load across all operating systems tested. The number of iterations chosen for the PHP test is 1,000,000 for all operating systems.

Next we will use the Build-Linux-Kernel test to further test the processor efficiency under a computationally complex instruction load. The Linux kernel is a large and widely used open source project that requires many parallel compilations to build. Kernel compilation is a good benchmark to use because the time taken to compile is heavily influenced by the operating system's organization [6].



Add	1213.98	1106.28	967.6	1200.97
Copy	1852.98	1748.5	1644.91	1830.65
Scale	1647.21	952.44	1199.33	1626.8
Triad	1200.97	650.13	970.65	1202.52
Average	1476.11	1115.26	1196.96	1453.68

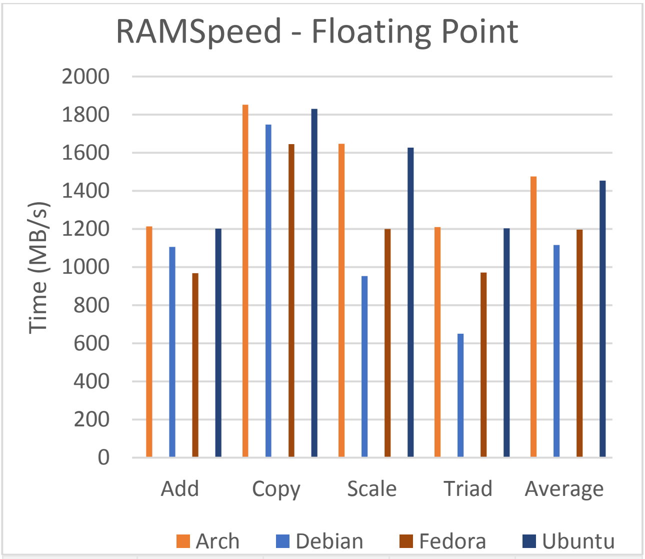
Figure 1. Integer Results of RAMSpeed benchmark

4. RESULTS AND ANALYSIS

4.1 Memory Results

RAMSpeed is a comprehensive memory cache and performance benchmark capable of running four tests: add, copy, scale, and triad in two modes: integer and floating point. These tests use synthetic simulations that correlate with real-world application memory usage. Figure 1 shows the results of the integer tests where the heights of the bars are measuring MB/S throughput, here a higher score is better.

Referring to the RAMSpeed integer results in Figure 1 we can see that Arch and Ubuntu produced better scores in all areas with an average of 1476.11(MB/S) and 1453.68(MB/S) respectively. Debian and Fedora split the integer tests with Debian performing better on the Add and Copy tests, and Fedora performing better on the Scale and Triad tests. The average scores for Debian and Fedora were similar, but significantly worse than Arch and Ubuntu with an average of 1115.26(MB/S) for Debian and 1196.96(MB/S).

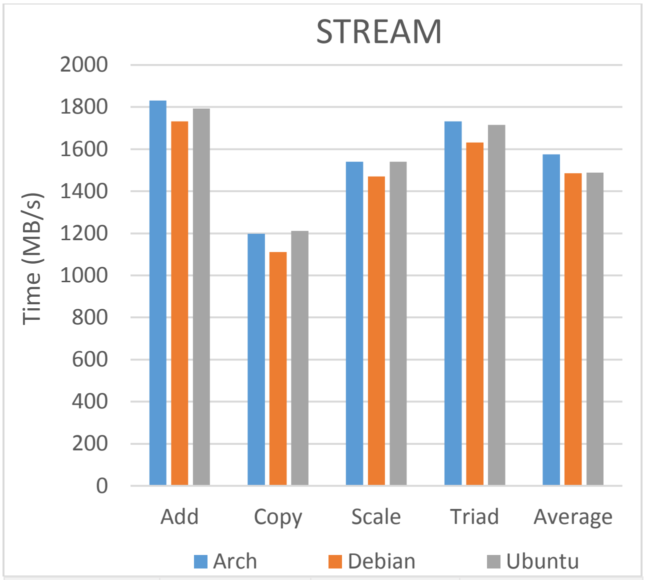


Add	1366.58	1251.63	1110.77	1348.38
Copy	1843.01	1753.22	1657.78	1847.02
Scale	1507.39	1336.1	1240.96	1499.9
Triad	1152.39	1056.07	906.88	1137.92
Average	1469.97	1348.1	1226.48	1460.42

Figure 2. Floating point results of RAMSpeed benchmark

The floating point results of the RAMSpeed benchmark are shown in Figure 2. Arch and Ubuntu again produce similar results, with an average throughput 1469.97(MB/S) and 1460.42(MB/S) respectively. Debian performed better in all testing areas with an average result of 1348.1(MB/S) than Fedora, which had an average of 1460.42(MB/S).

Additionally the STREAM benchmark was performed in order to verify the results of the RAMSpeed benchmark. However, STREAM could not be successfully run on Fedora. The STREAM results for the other operating systems did however correspond to the RAMSpeed results. Figure 4 shows the STREAM results for Arch, Debian, and Ubuntu. Here as in the RAMSpeed test the bars are measuring throughput in megabits per second, where higher bars indicate better performance.



Add	1830.95	1731.91	1792.2
Copy	1197.04	1110.33	1211.67
Scale	1647.21	952.44	1626.8
Triad	1200.97	650.13	1202.52
Average	1476.11	1115.26	1453.68

Figure 3. Incomplete results of STREAM benchmark

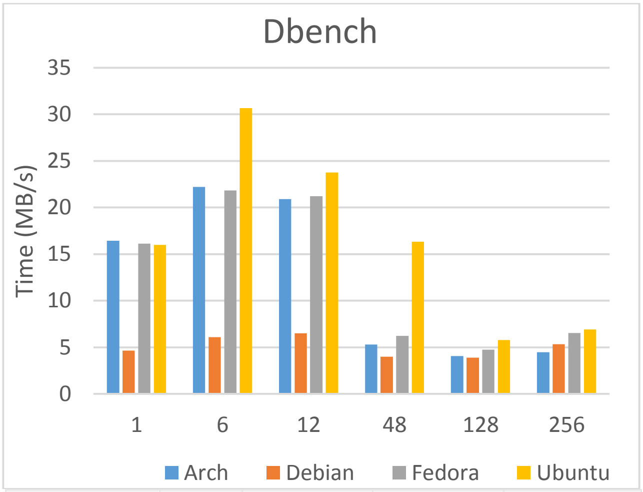
Figure 3 shows the incomplete STREAM results which are similar to the RAMSpeed test in that they indicate Arch and Ubuntu outperform Debian in memory performance. Although these results are incomplete they serve as confirmation that there are significant advantages in memory throughput on Arch an Ubuntu operating systems over Debain and Fedora.

4.3 Secondary Storage Results

The DBench test was used to simulate stress on the filesystem by running concurrent clients which generate I/O workloads. The number of clients range from 1 to 256. Once again we are measuring throughput in megabits per second. Referring to the Dbench results in Figure 4, the higher bars indicate better results. Intuitively, as more clients are added the performance degrades, however the rate at which performance degrades is important to consider. Debian produced the worst results out of the group in this test especially in the 1, 6, and 12 client tests, though, it performed better than Arch in the 256 client category. Ubuntu produced the best results, outperforming the other operating systems in all tests except one. It is important to note that in the 6, 12, and 48 client tests Ubuntu shows the slowest rate of throughput degradation. Overall, the operating systems produced similar results in the 128 and 256 client tests.

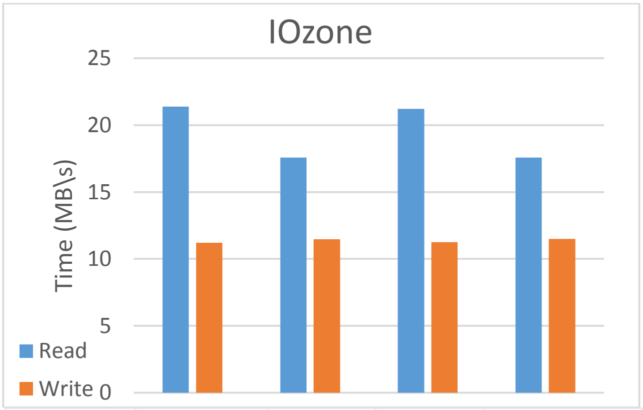
IOzone is a benchmark that measures read and write speeds on secondary storage. Figure 5 shows the results for both the read and write speed of each operating system in megabits per second. The read speeds were close for all operating systems, with Arch and Fedora slightly outperforming Debian and Ubuntu. The write speeds were virtually identical for all operating systems tested.

The results for the IOzone benchmark were too close to determine a significant performance advantage. However, the Dbench results indicated that Ubuntu exhibits superior secondary storage management. Conversely Debian greatly underperformed in this category.



1 Client	16.45	4.63	16.11	15.99
6 Clients	22.21	6.09	21.23	30.68
12 Clients	20.9	6.51	21.23	23.75
48 Clients	5.3	3.99	6.22	16.33
128 Clients	4.05	3.9	4.75	5.78
256 Clients	4.48	5.33	6.53	6.92

Figure 4. Results for Dbench benchmark



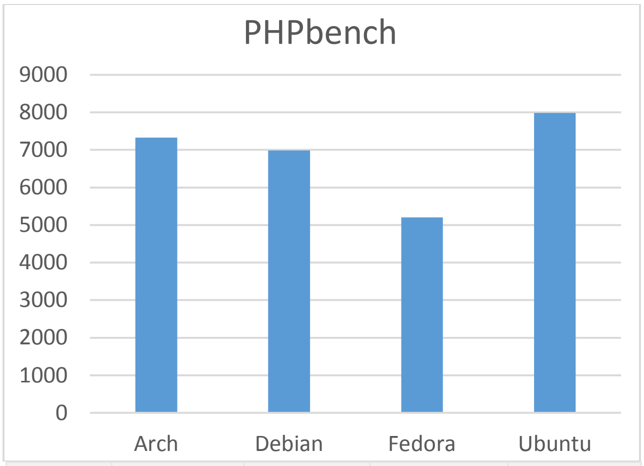
	Arch	Debian	Fedora	Ubuntu
Read	21.38	17.58	21.23	17.58
Write	11.2	11.48	11.25	11.5

Figure 5. Results for IOzone benchmark

4.4 Processor Results

The first test used to measure processor time management is PHPBench, a benchmark suite for PHP. PHPBench is a CPU intensive benchmark that runs 1,000,000 iterations of simple operations on the PHP interpreter. Figure 6 shows the results of the tests where the heights of the bars measure the total time in seconds to complete the test, the lower bars indicate better scores. Three trials were conducted on each operating system and averaged. The results were close for Arch and Debian with scores of 7327 and 6982 respectively. Fedora performed the best with a score of 5201, while Ubuntu, with a score of 7979, performed the worst. With a mean of 6872.25 and standard deviation of 1188.39, the results of

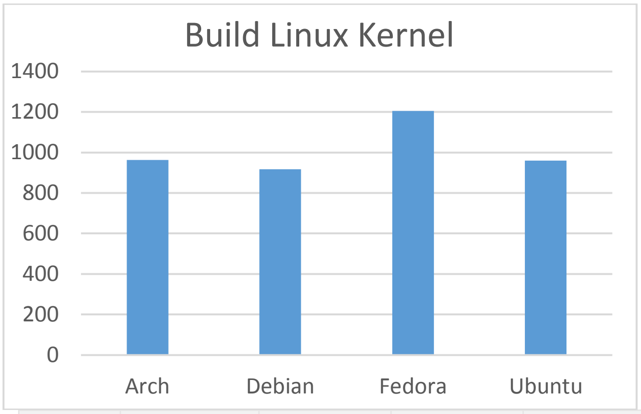
PHPBench showed that processor performance is sensitive to operating system configuration.



Score	962.5	915.75	1205.42	960.18

Figure 6. Results for PHPbench benchmark

The build Linux kernel benchmark simply tests how long it takes to compile the Linux kernel. This is an interesting benchmark for multi-processor because a large number of parallel compilations are required in order to build the kernel. As shown in Figure 7, it took approximately the same amount of time to build on Arch, Debian, and Ubuntu with respective scores of 962.4, 916.75, and 960.18. Fedora took the longest time to complete the build with 1205.42. These results indicate that Arch, Debian, and Ubuntu performed equally well on the test, while Fedora performed significantly worse.



Score	7327	6982	5201	7979

Figure 7. Results for Build Linux Kernel benchmark

Comparing the two processor tests reveals an interesting discrepancy. While Fedora performed better than the other operating systems during on PHPBench, it performed the worst on the Build Linux test. Arch, Debian, and Ubuntu each performed about the same on both tests relative to the other operating systems. This implies that the disparity between the results could be an indication of software optimization for certain types of processor operations on Fedora Linux.

5. Conclusion

The primary goal of this research is to compare the system performance of Linux operating systems on a low powered device. Arch, Debian, Fedora, and Ubuntu are four popular Linux distributions that were chosen for comparison in three main areas of operating system function: processor, memory, and secondary storage management. Comparisons were made using popular open-source benchmarking tools available from the Phoronix Test Suite. The results of the processor tests showed no clear winner, as each operating system produced similar results. In the memory management tests Arch and Ubuntu outperformed Debian and Fedora in both integer and floating point operations. Lastly Ubuntu showed superior disk I/O times in the multi-threaded DBench test, while all operating systems performed similar in the IOzone read/write test. No one operating system clearly outperformed the others in all areas, leading to the rejection of hypothesis that Arch Linux would produce better processor, memory, and disk results.

REFERENCES

[1] “ARM Ports.” 9 Jan 2016. Software in the Public Interest, Inc. https://www.debian.org/ports/arm/ Accessed: Jan 27 2016.

[2] “Develop Windows 10 IoT apps on Raspberry Pi 2 and Arduino.” 2015 Microsoft Corporation https://dev.windows.com/en-us/iot Accessed: 27 Jan 2016.

[3] Dinndorf J., “Performance Comparisons of Operating Systems for the Raspberry Pi”, The 14th Winona Computer Science Undergraduate Research Symposium, Winona State University, 2014, pp. 11-17. http://cs.winona.edu

[4] “Fedora ARM.” 2015. Red Hat, Inc. https://arm.fedoraproject.org/ Accessed: Jan 27 2016.

[5] “FreeBSD/ARM Project.” 14 April, 2015. FreeBSD.org. https://www.freebsd.org/platforms/arm.html Accessed: 27 Jan 2016.

[6] Hatt N., Sivitz A, and Kuperman B. A., “Benchmarking Operating Systesms”, Conference for Undergraduate Research in Computer Science and Mathematics, Oberlin College, November 2007.

[7] Hollander R. M. and Bolotoff P. V., “RAMspeed, a cache and memory benchmark”, Alisar Enterprises, http://alasir.com/software/ramspeed/, Accessed: 8 March 2016.

[8] Lencse G. and Répás S., “Method for benchmarking single board computers for building a mini supercomputer for simulation of telecommunication systems”, in. Proc. Int. Conf. on Telecomm. and Signal Processing (TSP 2015), Prague, 2015, pp. 246–251.

[9] Monk, S. (2015). Programming the Raspberry Pi: Getting Started with Python (2nd ed.). New York: McGraw-Hill.

[10] Norcott W. D., “IOzone Filesystem Benchmark”, http://www.iozone.org/, Updated: 23 January 2016, Accessed: 8 March 2016.

[11] “Open-Source Benchmarking”, phoronix-test-suite.com, http://www.phoronix-test-suite.com, Accessed 21 January 2016.

[12] “Raspberry Pi 2.” 2015. ArchLinuxARM.org http://archlinuxarm.org/platforms/armv7/broadcom/raspberry-pi-2 Accessed: Jan 27 2016.

[13] “Raspberry Pi 2 model B” February 2015 RaspberryPi.org http://www.raspberrypi.org/raspberry-pi-2-model-b/ Accessed: 15 March 2014

[14] Silberschatz, A., Galvin, P. B., & Gagne, G. (2005). Operating system concepts (7th ed.). Hoboken, NJ: J. Wiley & Sons.

[15] “Speed Class.” SD Association. https://www.sdcard.org/developers/overview/speed_class/ Accessed: 29 Jan 2016.

[16] Tridgell A. and Sahlberg R. “DBench”, Samba, https://dbench.samba.org/web/index.html, Accessed 8 March 2016.

[17] “Ubuntu for ARM.” 2016. Canonical Ltd. http://www.ubuntu.com/download/server/arm Accessed: 27 Jan 2016.

Mesh Sensor Network for Atmospheric and Weather Data Acquisition

Nicholas McNeely
Winona State University
Winona Minnesota
nmcneely11@winona.edu

ABSTRACT

As a campus concerned with sustainability, tracking the atmospheric and weather conditions on the campus is essential to informing and educating students about their environment. This project investigated a way to gather weather and atmospheric data using a mesh sensor network. In this application the mesh network must fulfill the requirements of reliability, minimal latency, minimal bandwidth usage, and minimal power consumption. The network consisted of individual nodes and a central node or base station. Each of these nodes were designed gather sunlight, humidity, and temperature data. The base station gathers the same data as well as information about ozone levels. The base station also was designed to act as a relay to move the gathered data off of the mesh network and into an Internet accessible database. The system was tested using three nodes and a base station. Each node was one hundred to two hundred feet from either the base station or another node. In this configuration the system was unable to fulfill the requirements. Further research into alternative configuration is needed.

General Terms

Management, Measurement, Performance, Design, Reliability, Experimentations.

Keywords

Mesh Network, Mesh Sensor Network, Wireless, Solar Powered, Self-Sustaining, Sensors, Sensor Node, Weather Data, Atmospheric Data

1. INTRODUCTION

Winona State University is dedicated to being environmentally conscious. The university has programs that improve the sustainability of student housing, dining services and university transportation [1]. Though the university as a whole may be environmentally conscious and concerned, it is important that current and new students be educated and made aware of sustainable practices and programs. To help in this effort, the chemistry, physics, and art departments have begun a project to increase awareness about the air quality on the campus. They proposed and have begun work on a sculpture and accompanying electronics to monitor ground level ozone and display those levels in an informative and artistic way [2]. This project sparked interest in not only monitoring ozone levels, but also in monitoring other environmental data such as temperature, humidity, and ultraviolet/sun light level.

One way to educate students about sustainability and the environment is to engage them with a creative display detailing information about the environment where they live, work, and attend school. The ozone project understood this. They created an engaging piece of eye catching art intended to draw your attention and then educate you about ozone levels. Because they already had the sculpture in place, it was decided that augmentation to the existing system would be the best way to bring attention to the additional environmental data. Thus, the goal for this project was to develop a system to collect temperature, humidity, and sunlight data and transmit it to a small computer located at the sculpture. This data could then be incorporated into the ozone project's display at a later time.

1.1 System Constraints

The first step was to decide how, what, and where the additional data would be gathered. The question about where the data would be gathered influenced strongly the other two decisions. First of all, data was to be gathered from multiple points across the campus to allow for averaging differences in readings. Secondly, large installations were not feasible due to cost and space availability. Thirdly, in keeping with the sustainability principle these new data gathering stations needed to be solar powered. This meant they needed to be in a position that allowed them to get as much sunlight as possible. This also meant that the stations had to be small enough to be attached to the tops or sides of buildings or affixed to something like a light pole.

Due to the limitation in size, the stations would only be able to generate a small amount of power. This limited the types of sensors and equipment that could be used to gather the data. Three data types were selected, ultraviolet/sunlight, temperature, and humidity.

Thus, the data was to be gathered as follows: a small station that was solar powered, using three small low power sensors to gather the ultraviolet/sunlight sensor, a temperature sensor, and a humidity sensor. These would be controlled by a low power controller that was paired with a small low power wireless transceiver.

Proceedings of the 16th Winona Computer Science Undergraduate Research Seminar, April 27, 2016, Winona, MN, US.

1.2 Mesh Network

Wireless technologies allow for flexible device placement. Wireless communication comes with its own problem though: power requirements. Wireless technologies like Wi-Fi and cellular radios require massive amounts of power. This meant that the existing Wi-Fi infrastructure on the campus was infeasible. Low power point to point transceivers provided a viable alternative.

The difficulty with lower power receivers is the distance at which they can communicate. For instance a sensor station on one side of the campus would not be able to communicate with one on the other side of the campus. One way to resolve this is to use a network organization scheme known as mesh networking. In a mesh network every transceiver acts a relay, and every node is not necessarily connected to every other node. For instance consider Figure 1. In order to get a message to node C, node A has to find a node or a series of nodes that can reach C. In Figure 1 node A can reach node B, and node B can reach node C. So if node A gives the message to node B, node B can then pass that message on to node C.

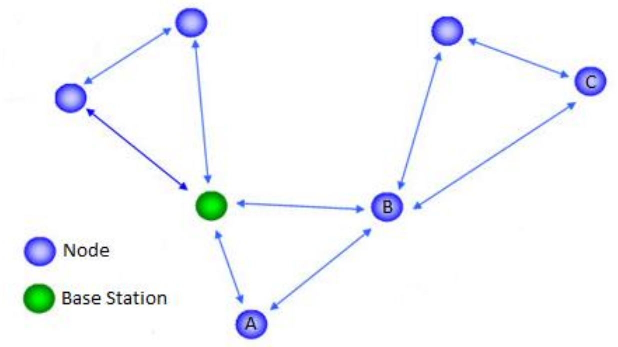


Figure 1

Through the use of a mesh network configuration, low power radios can be used to send messages over distances that would normally not be achievable due to their limited range. In this research a wireless mesh network system was built to provide a reliable way to aggregate data gathered on a set of distributed nodes. The reliability of such a mesh network was investigated by examining its ability to provide consistent service.

2. METHODLOGY

2.1 Materials

In order to investigate the reliability of a mesh network as a system for gathering weather and atmospheric data, a prototype system was necessary. It was decided that a custom built system would be used for two reasons. Using a custom system allows for an exact understanding of how the system components would affect the results. In addition, a custom system allows for direct programmatic control enabling modifications to improve the system performance according to the environment. The design and specifications of this system are described below.

2.1.1 Nodes

Each node is composed of four major components, a controller, a wireless module, a set of sensors, and a solar panel based power source. These components are either placed inside or affixed to the exterior of a hard-plastic, water tight enclosure. When deployed these nodes are placed outdoors in an area where they receive optimal sunlight for power generation.

The controller chosen was an Adafruit Feather M0 Basic Proto - ATSAMD21 Cortex M0. This controller was an ideal solution because it possessed the necessary sensor interfaces, was compatible with the selected wireless module, and had the necessary circuitry for charging a battery using a solar panel.

There were a number of challenges that had to be addressed while selecting a wireless module. Power consumption and operating frequency were the most challenging. Originally the plan had been to use a module called the ESP8266. This module operated using Wi-Fi in the 2.4 GHz band. After consideration this module was ruled out because it could consume as much as 170ma. Also the network administrators in the area where these nodes would be deployed did not want devices broadcasting in the 2.4 GHz band. The second module considered was an XBee Pro 900. This module solved the operating frequency dilemma. It operated in the 900 MHz band, one that the network administrators were not using and thus did not cause any problems. However, it made the power problem even worse. This second module could consume as much as 210ma.

The third module considered, and the one selected for use, was a HopeRF RFM22B-S2. This module met the needs of the project, using only 87ma and operating the 900 MHz band.

Three sensors were selected: a temperature sensor, a humidity sensor, and an ultraviolet/light sensor. These sensors were selected to gather basic weather data and atmospheric data. The specific sensors – the Waterproof DS18B20 from Adafruit, the DHT11 from Adafruit, and the SI1145 from Adafruit – were selected because of their ease of integration with the selected controller.

The power component of each node is a combination of the charging circuitry built into the controller, a 2200mAh lithium-polymer battery, and a 5V 2.5W solar panel. This combination allows the solar panel to charge the battery when it is receiving enough sunlight. The battery allows the system to run at night and on days when there isn't enough sunlight to power the system.

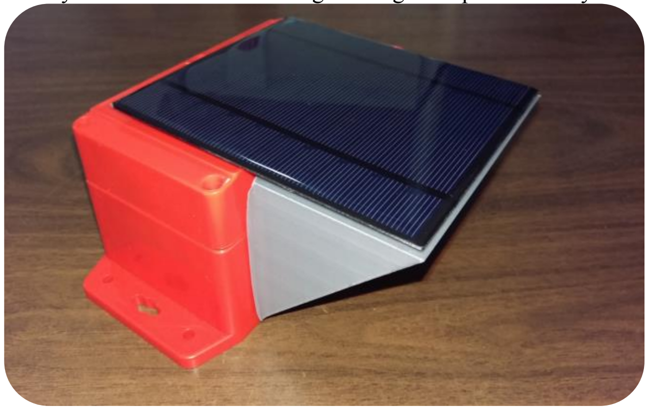


Figure 2 Assembled Sensor Box

The components were assembled using a hard-plastic, water-tight sensor box and two custom 3D printed parts. The sensitive controller and wireless module components, were sealed inside the sensor box. The solar panel is affixed to the top of the sensor box. The light sensor is attached to the sensor box using one of the custom 3D printed parts. The humidity sensor is located inside a large grey plastic wedge. The waterproof temperature sensor is also located inside the 3D printed wedge. Each of the components outside of the sensor box are fed back to the controller through a series of sealed ports in the sensor box. Figure 2 shows an assembled sensor box. The red box is the water-tight portion. The grey wedge houses the sensors and the solar panel on top feeds power into the system through the lid of the box.

2.1.2 Base Station

The base station is a modified version of a sensor node. In addition to acting as a sensor node the base station also acts as a relay, moving data from the mesh network into a system that is able to put it in an Internet accessible database. The controller and the wireless module are the same ones that are used in sensor nodes. It is also important to note how the base station is powered.

The base station is part of the collaborative ozone data sculpture. Because it is part of the sculpture it made sense from a design point to use the power that the sculpture generates rather than having the base station generate its own power. The power is drawn from the sculptures batteries that are charged using a much larger 180W solar array. This means that power is not a major concern for the base station.

2.1.3 Software

All of the code for the base nodes and base station was writing in C using the Arduino IDE and compiled using the Adafruit SAMD Boards profile. There are two core aspects to the software used to run the nodes and base station. The first is the code that handles the wireless communication. The second is the code that handles the sensors on each of the nodes. The sensors and the wireless module each have a respective Arduino library that simplifies the process of writing code for the application that uses them.

The heart of the code that controls the wireless communication is the *RadioHead Packet Radio library for embedded microprocessors* [3]. This library provides the code necessary for networking the nodes and base station in a mesh network configuration. Once the library is configured, sending a message is only a matter of calling the function sendToWait() and passing the data to be sent, the size of the data, and the address of the destination node.

To gather the data from the sensors, the Arduino libraries are used. They are the Adafruit ST1145 library [4], the Adafruit DHT library [5], and the Dallas Temperature library [6]. Each of these libraries provide the necessary functions for acquiring data from their respective sensors. Additionally an additional function for acquiring an analog voltage reading is used to determine the battery level.

On each of the nodes, code is executed in a repeating loop that follows the sequence in Figure 3.

1. Configure wireless module library
2. Configure sensor libraries
3. Read data from each sensor and store the result
4. Using the stored values build the message to be sent to the base station
5. Send the message to the base station
6. Pause execution for a set amount of time to save power
7. Repeat starting at step 3

Figure 3 Node Code Loop

This series of steps executes very quickly, completing in under two seconds. This allow the node to be in a more power efficient state. Because the base station acts as a relay as well as a sensor node the execution loop is slightly more complex. The sequence of steps that the base station executes can be seen in Figure 4.

2.2 Procedure

Nodes were to be placed in their locations on campus, approximately one hundred to two hundred feet apart with at least one node within that distance of the base station. The base station was to be connected to the sculpture's electronics and power source. Everything was to be connected, the nodes would begin transmitting the data to the base station. The base station was to receive the data and relay it to the sculpture which would log it in a database. The data would be logged in the database, a message number corresponding to the number of times the node had transmitted data, the readings from each sensor, and the battery voltage at the time of each message.

1. Configure wireless module library
2. Configure the sensor libraries
3. Set aside space for incoming messages
4. Check if message is received if not go to set 5
5. If message is received send it to the serial port
6. Repeat starting at set 3
7. Once every set amount of time read data from each sensor and send it to the serial port.

Figure 4 Base Station Code Loop

In order to establish the reliability of mesh network system, three pieces of information were going to be examined. First, the number of times the message number resets to the initial value of one was examined. A reset to the initial value indicates that the system was either forced to restart or had to shut down for some time. Second, the message number was examined to determine if a significant amount of data was lost. Third, the battery voltage was tracked to determine if there was a regular loss of power or to see if there was a long term downward trend.

3. RESULTS AND ANALYSIS

The base station and nodes were placed in testing configuration, emulating the configuration they were to be placed in on the campus, the power sources were connected, and each was turned on. The output form the base station was monitored, but no data was being received. All of the nodes and the base station were brought back to the lab for diagnostics. The nodes and base station were powered on and everything appeared to be working. The nodes were transmitting data and the base station was receiving and relaying the data back to the computer.

Upon further investigation and examination of the data it was found that the signal strength between the nodes and the base station was very weak. The RSSI (Received Signal Strength Indicator), in the lab with the nodes and base station only a few feet apart, was between -70 and -80. Preliminary tests had shown RSSI at approximately two hundred feet to be between -65 and -75. Since high values are stronger signals the current RSSI values indicated something was causing a severe decrease in signal strength from what was measured in the preliminary tests. Three aspects of the design were examined in an attempt to determine what had caused the degradation in signal strength.

The first attempt to diagnose the problem was to move the base station and the nodes to a new environment to ensure that there was no localized signal pollution. The nodes and base station were taken to a residential neighborhood rather than the university campus. This test showed no improvement in signal strength.

After signal pollution was ruled out, the electrical integrity of the connection between the wireless module and its antenna was examined for problems. This was done by directly attaching the antenna to the wireless module rather than having it connected through the circuit board. This did not lead to an improvement in performance.

After the connection to the antenna was determined to be intact a new antenna was tested. Using a more precise measuring tool and a solid copper wire, a new antenna was fashioned. This new antenna replaced the old antenna. The signal strength was measured again, and found to have no improvement.

These results were not expected. They show that in the current configuration these modules are not suitable for this system. This avenue of inquiry should not be abandoned though. Preliminary results showed that power consumption of this system was within the required margins for self-sufficiency.

4. CONCLUSION

It is the conclusion of this report that this particular configuration of hardware is not a suitable system for gathering weather and atmospheric data. The radio chosen for this system failed to meet the requirements of the specification. It is believed that this failure was the result of a failure in the electrical design of the system. As such, two modified systems may prove to be more viable and should be the focus of further study.

The first of these systems would use the same components as the current design. The difference would be a strong focus on improving the electrical design. Instead of using a general purpose development board, a high quality application specific PCB should be designed for the system.

The second suggested system would replace the current radio module with a module that is more self-sufficient. This module would remove the need to focus on difficulties of wireless electronics.

5. ACKNOLWEDGEMENTS

I thank my advisor Dr. Joan Francioni for her guidance throughout the project. I would like to thank the Computer Science Department at Winona State University for funding my research. I also thank Dr. Nathan Moore for his comments and suggestions with respect to the electronics and the Art department at Winona State University for assistance in 3D printing parts for the sensor boxes.

6. REFERENCES

[1] Winona State University, "Green Campus: Sustainability in Practice," [Online]. Available: http://www.winona.edu/green/campus.asp. [Accessed 28 1 2016].

[2] C. L. M. F. O. Jeanne Franz, "What's the Air Quality Like Today? Development of an Interactive Display to Measure Ground Level Ozone and Educate the Community about Renewable Energy," Winona, 2016.

[3] M. McCauley, "RadioHead Packet Radio library for embeded microprocessors," 12 4 2016. [Online]. Available: http://www.airspayce.com/mikem/arduino/RadioHead/index.html. [Accessed 19 4 2016].

[4] L. Ada, "Adafruit SI1145 Breakout Board - UV index / IR / Visible Sensor," Adafruit, 6 2 2016. [Online]. Available: https://learn.adafruit.com/adafruit-si1145-breakout-board-uv-ir-visible-sensor/overview. [Accessed 19 4 2016].

[5] L. Ada, "DHTxx Sensors," Adafruit, 4 5 2015. [Online]. Available: https://learn.adafruit.com/dht/overview. [Accessed 19 4 2016].

[6] M. Burton, "Dallas Temperature Control Library," 15 1 2016. [Online]. Available: http://milesburton.com/Main_Page?title=Dallas_Temperature_Control_Library. [Accessed 19 4 2016].

Configuring a Drone Flight Controller over WiFi

Mitchell Gerber
507-273-6363
mgerber11@winona.edu

ABSTRACT

The Naze32 drone flight controller is a popular choice today among hobbyists for building their own drones. The Naze32 must be configured with a computer, which can sometimes be difficult to access when out in the field operating the drone. A method has been developed to configure the Naze32 flight controller wirelessly over a WiFi network using an ESP8266, which is a microcontroller that is cheap, small in scale, and capable of WiFi communication. The ESP8266 physically connects to the Naze32 and communicates with it using the serial communication protocol. The ESP8266 then starts a WiFi access point and serves a dynamic web interface, in which users can pass configuration settings to the Naze32. Because the ESP8266 is limited in space and processing power, testing has been done to determine what features the web configuration tool is capable of offering. This system has been proven to allow more effective and accessible configuration of the drone flight controller without the use of a desktop computer.

Keywords

Naze32, ESP8266, WiFi, Serial Communication.

1. INTRODUCTION

Multirotors, which are commonly referred to as "drones", have become extremely popular within the past few years due to technological advancements in the software and hardware within them. The Naze32 is a popular flight controller among the multirotor racing community because it is cheap, powerful, and uses open source firmware. Configuration of the Naze32 requires the Cleanflight Configurator, which is an open source configuration tool for the Naze32 that can only run on a desktop computer [4]. The problem is that a computer is required to configure settings on the Naze32 flight controller. This may limit the user if they are out in the field operating their drone. A solution is developed to configure the Naze32 flight controller over WiFi. The ESP8266 is used to provide WiFi configuration of the Naze32 flight controller because it is cheap, capable of WiFi communication, and can run its own web server.

Arduino is a company that manufactures open-source hardware and software. They create microcontrollers that are cheap and easily programmed with their Arduino Integrated Development Environment (IDE). A few versions of the Arduino microcontrollers were considered for the development of this project. The main problem is that adding WiFi capability to an Arduino can be costly. The ESP8266 is an inexpensive microcontroller similar to the Arduino, but has WiFi capabilities built in. The ESP8266 is becoming a popular alternative to the Arduino because it is cheap, more powerful, and has built in WiFi functionality. There are different versions of the ESP8266, but they are all very similar. The only differences are the size of flash memory and board layout. For this project I used the ESP-12e version of the ESP8266. This microcontroller is developed by a company called Espressif and they have publicly released the SDK(Software Development Kit). The SDK has been used to create a development environment for the Arduino IDE called ESP8266 Arduino. ESP8266 Arduino provides many of the Arduino libraries, which make programming the ESP8266 quicker and simpler than using the SDK [2].

Many hobbyists choose the Naze32 as their primary flight controller because it is cheap, powerful, modular, and is easily configurable with a computer. Although it is easily configurable, users are still limited in the sense that they have to have access to a computer in order to configure or change settings on the flight controller. Incorporating the ESP8266 into the configuration process of the Naze32 makes it accessible by a wider array of devices. This means that the user could be out in the field operating their drone, stop and plug the ESP8266 into the flight controller, edit configurations and then save and continue where they left off. Although a computer is still needed for initial configuration, this eliminates the use of a computer for quick adjustments.

Space on the ESP8266 is limited, which means only the most important features of the Cleanflight Configurator were focused on. These features include adjustments that affect flight characteristics of the drone. There are multiple stages of building a drone, which include assembly, calibration, initial setup, and tuning. The tuning stage is arguably the most important part of building a drone that flies well. This stage requires the user to observe flight characteristics after making adjustments. This can be a tedious process because the user has to plug their drone into a computer after every flight if they want to make adjustments. This is made quicker and easier with the implementation of WiFi configuration.

The settings on the Naze32 that affect flight characteristics are known as the PID (Proportional Integral Derivative) settings. A PID controller is a control loop feedback mechanism that has been used for many years in industrial control systems [6]. This system is also used in multirotors and helps it fly smoothly. A drone with PID settings out of tune will experience oscillations. Depending on how out of tune the PID settings are, these oscillations may be clearly visible. Another setting on the Naze32 that affects flight characteristics is the loop time. The loop time is the time it takes to complete a control loop during flight. This includes sensor measurements, data processing, and PID calculations [5]. Adjusting the loop time and PID settings on the drone flight controller are the important features are implemented in the user interface.

Proceedings of the 16thWinona Computer Science Undergraduate Research Seminar, April 27, 2016, Winona, MN, US.

2. Hypothesis/Question

Features of the Cleanflight Configurator can be implemented on the ESP8266 microcontroller, which can then be used to configure the Naze32 flight controller wirelessly over a WiFi network. Can a responsive web interface be developed that is consistent with the Cleanflight Configurator that will fit on the ESP8266? This paper and research explores the possibilities of the ESP8266 with respect to serial communication, serving dynamic web pages, and WiFi communication.

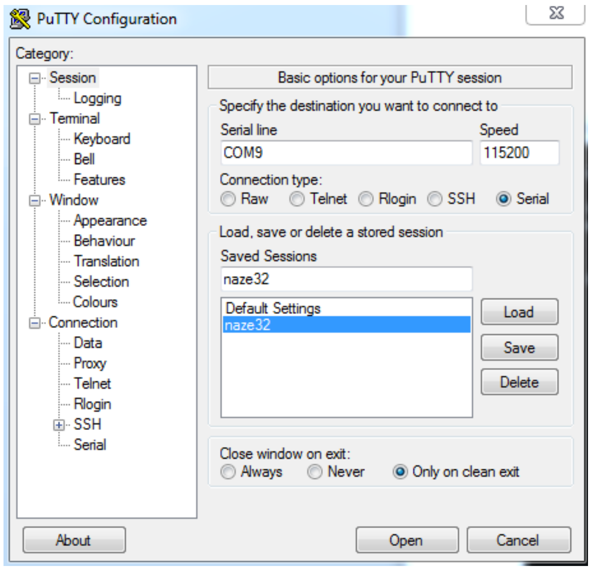
3. Method

3.1 Development Method

This project followed the agile software development model. Development was divided up into many different parts and different languages such as C, Javascript, HTML, and CSS were used as well as different development environments. Each of these parts has been worked on individually and in no particular order. There were certain objectives and goals throughout the development process that are explained in depth in the method section. The agile software development model is an appropriate model to follow and has positively impacted the implementation of this project.

3.2 Communicating with the Naze32

The first initial step is to establish communication between the ESP8266 and Naze32. Both of these microcontrollers communicate using the serial communication protocol, which means that they can communicate directly. The Naze32 can be configured directly with CLI commands issued to it via serial at a baud rate of 115,200 Bd. To test this out, a computer is used to configure the Naze32 with a serial to USB adapter and a program called PuTTY. PuTTY allows Windows users to issue serial commands to a connected device. As displayed in Figure 1, a communication port can be selected (COM9) as well as the baud rate for communication speed (115200 Bd).



Specify the destination you want to connect to

Serial line: COM9
Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

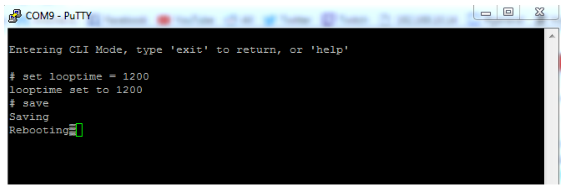
Saved Sessions: naze32

Default Settings
[✓ naze32]

Close window on exit: Always Never [✓ Only on clean exit]

Figure 1. PuTTY to connect with Naze32

Figure 2 displays an open serial communication terminal with the Naze32 using Putty. The “#” command is issued to the Naze32, which causes it to enter the CLI mode. The CLI mode allows the Naze32 to accept various configuration commands [1]. The example shown above displays the process to change the loop time of the Naze32. The “set” command followed by the “save” command causes the Naze32 to execute the command and then reboot.



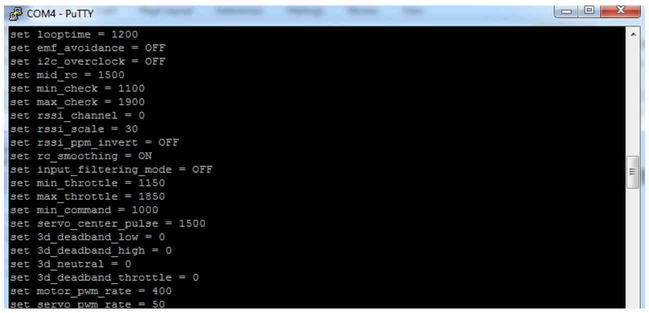
```
Entering CLI Mode, type 'exit' to return, or 'help'

# set looptime = 1200
looptime set to 1200
# save
Saving
Rebooting
```

Figure 2. CLI mode is entered

3.3 Parsing Naze32 Response

The Arduino IDE (Integrated Development Environment) was used to program the ESP8266 in the C language. The Arduino IDE offers many useful libraries that will be helpful in developing the back end functionality of the WiFi configuration tool [2]. The ESP8266 serves the purpose of sending commands to the Naze32 flight controller and receiving and parsing responses. All Naze32 settings can be received by issuing the command “dump”. As shown in Figure 3, Naze32 settings are received in the format “set <parameter> = <value>”. This makes it easy to process the entire string and store each value. A subset of these settings needs to be displayed on the client side and there are multiple ways of achieving this.



```
set looptime = 1200
set emf_avoidance = OFF
set i2c_overclock = OFF
set mid_rc = 1500
set min_check = 1100
set max_check = 1900
set rssi_channel = 0
set rssi_scale = 30
set rssi_ppm_invert = OFF
set rc_smoothing = ON
set input_filtering_mode = OFF
set min_throttle = 1150
set max_throttle = 1850
set min_command = 1000
set servo_center_pulse = 1500
set 3d_deadband_low = 0
set 3d_deadband_high = 0
set 3d_neutral = 0
set 3d_deadband_throttle = 0
set motor_pwm_rate = 400
set servo_pwm_rate = 50
```

Figure 3. Part of the response from issuing command "dump" to Naze32

Javascript is used on the client side, therefore the most practical thing to do is store settings in JSON (Javascript Object Notation) format. There are a few different approaches to this that were considered. One option is to dump the settings into one giant string and send it over to the client side to be formatted. This would be a good option if the ESP8266 was completely starved of resources because it would reduce the amount of processing on the server side. The next option is to use a JSON library for the ESP8266. This seemed like the most practical approach to solving the problem and would be best for a production environment due to reusability. The downside of this option is that it requires extra overhead to use the entire library when in reality only a small portion is needed. Although some JSON libraries are less than 5kb, keeping the size of the entire program to a minimum is a must. This left a final option of just creating a new string that is in JSON format. This was appealing because it did not use any additional libraries and the processing was still done on the server in order to deliver a JSON string through the proper use of an API. This was the option that was chosen and although it may not have been the best in terms of reusability or speed, it was sufficient for this implementation.

3.4 Developing the User Interface

The user interface is intended to only allow the user to adjust settings which are the most important during the tuning phase. PID and loop time are the configuration settings that are adjusted the most when out in the field operating the drone, therefore the user interface was limited to these to prevent a cluttered user experience. Although these are the only settings included in the user interface, there is a section that allows users to input CLI (command line interface) commands. This provides the option of adjusting any configuration setting within the Naze32 flight controller. Figure 4 displays part of the user interface. The user interface was developed with simplicity as a priority. Each setting can be adjusted by a plus and minus button, or the user can enter in a custom value.

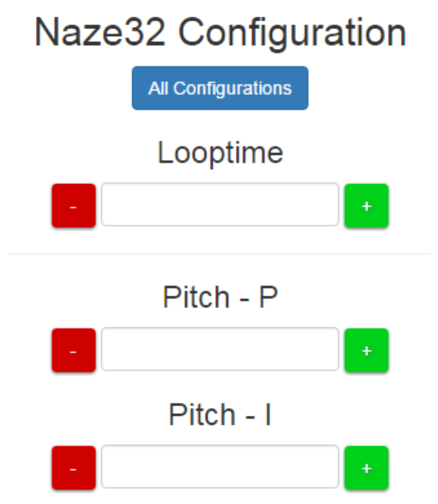


Figure 4. A section of the user interface.

There are a few different ways to serve the user interface from the ESP8266. The first way is to use the ESP8266's file system. This method is nice because files can be saved as actual HTML files. The ESP8266 can receive an HTTP request, open up the requested HTML file, and serve it to the user just like any normal web server. The only problem with this method is that is makes the code a little more complex and it also requires the flash memory to be written to whenever an update to the HTML document occurs. This can be a tedious task during the development phase. Because of this, another option was chosen to serve the static HTML files to the user. The ESP8266 can serve web pages in a character array, which can be stores in flash memory. To prevent making the main program cluttered, this is stored in a separate header file. C allows an escape sequence, which was very useful for storing the entire HTML file without escaping every special character. The sequence "R"=====()=====";" is used to escape every special character by storing the HTML document within the two parenthesis.

The ESP8266 does not technically serve any dynamic web pages in this case, therefore dynamic functionality is handled on the client side using Javascript. When the user accesses the index page the Javascript is triggered, which sends an HTTP request to the ESP8266's restful API to obtain the configuration settings and fill in the rest of the web page. Each field can then be adjusted with the plus and minus buttons or by an input value. Each one of these values gets added to the parameter list for when the user submits the configuration. Now, only the changed configurations are passed to the ESP8266, which then get sent over serial to the Naze32 to be saved.

3.5 Testing

The user interface developed during this project is expected to be used across a wide array of devices varying in screen resolutions. Because of this, a responsive design was emphasized. The Google Chrome web browser offers a device simulator for testing responsive websites as different devices. This was used to test the user interface for its responsiveness.

There are different versions of the ESP8266 and for this project the ESP-12e is used. This specific version has a flash memory size of 4mb, which is the largest flash size currently available for the ESP8266. This is more than enough space for a program that serves one HTML page along with some CSS and Javascript. The content being served is all stored in a character array in flash memory. The current user interface design only takes up 8kb, which is very small and is easily served by the ESP8266. CSS and Javascript are both included in the HTML document. Although this is not normally good practice, it lessens the amount of requests the server has to process. Keeping the amount of requests to the server to a minimum will keep the user experience running as smoothly as possible. This ultimately increases the speed at which the initial web page loads. Testing has been done to see how the ESP8266 reacts when serving different sized web pages.

4. Results

Development and testing has proven that features of the Cleanflight Configurator can be implemented within a web interface served by the ESP8266 microcontroller, which can then be used to configure the Naze32 flight controller wirelessly using any WiFi enabled device. As shown in Figure 5, users are able to physically connect the ESP8266 to the Naze32 flight controller. The ESP8266 then starts an access point and serves a dynamic web interface. Users can then connect to the access point and access this page, which provides a subset of Naze32 configurations from the Cleanflight Configurator. The user interface also provides manual CLI input commands for more advanced users. User input is then manipulated on the ESP8266, passed to the Naze32 using serial communication, and then saved. The new configuration settings are then passed back to the ESP8266 where they are displayed on the web interface for the user to observe.

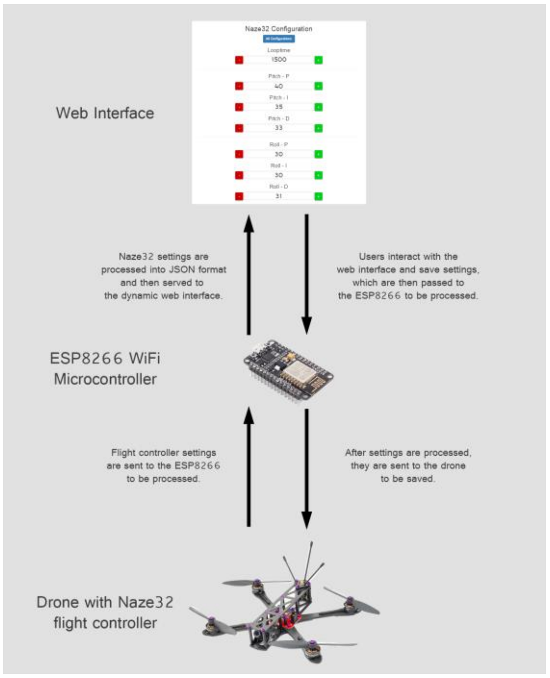

Naze32 Configuration
Web Interface
Naze32 settings are processed into JSON format and then served to the dynamic web interface.
Users interact with the web interface and save settings, which are then passed to the ESP8266 to be processed.
ESP8266 WiFi Microcontroller
Flight controller settings are sent to the ESP8266 to be processed.
After settings are processed, they are sent to the drone to be saved.
Drone with Naze32 flight controller

Figure 5. Illustration of the configuration process.

Testing was done to examine the behavior of the ESP8266 for different sized documents. Through trial and error it was concluded that any file over 35kb did not load in the browser. An issue like this is hard to debug because the ESP8266 does not produce any error codes for this specific problem. A very large character array is used, therefore it is assumed that an overflow may be occurring, which causes the blank page to load. The user interface developed in this project is around 8kb, which is easily served by the ESP8266.

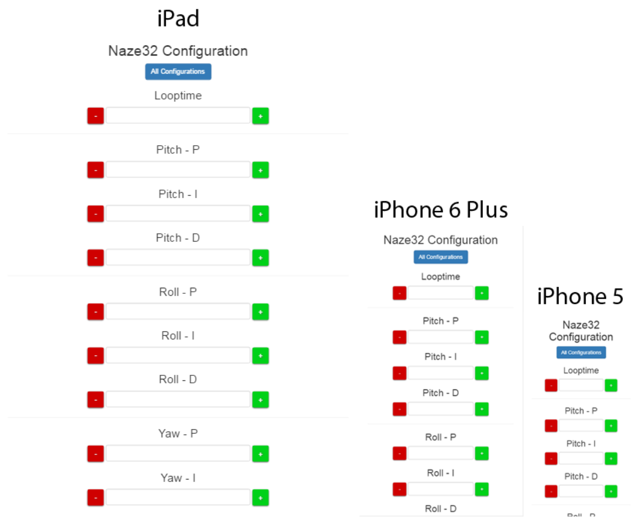


Figure 6. Responsive web design.

After testing the user interface with the Google Chrome device simulator it is concluded that it is consistent across many different devices. As shown in figure 6, the user interface adapts to different screen sizes. This allows the user to connect and configure their drone without worrying about which device they are using.

5. Conclusion

Throughout development and research there are many things that have been learned. It is possible to implement WiFi configuration into the Naze32 drone flight controller using the ESP8266 WiFi microcontroller. There are different methods for achieving this, but serving web pages from a microcontroller has its limitations. The method that was used in this project only allows files of less than 35kb to be served by the ESP8266. There are ways around this, but the user interface size was kept less than 35kb. The small size of the program allowed the web server to run smoothly and provided a robust interface for the user.

5.1 Future Development

This entire project only scratched the surface of the possibilities for the configuration of drones with the ESP8266 microcontroller. For example the Naze32 flight controller supports a live telemetry data stream through the same serial connection that was used in this project. Future development and experimentation could include parsing through the telemetry data stream and displaying it on the interface along with the configuration settings. This would allow the user to see a live stream of data such as RSSI, battery voltage, and flight time. Other future work may also include testing the limits of the ESP8266's wireless communication. What is the range of the ESP8266? Does the 2.4ghz signal interfere with the drone transmitter and receiver? These are some questions that could be answered with more testing in the future.

6. ACKNOWLEDGMENTS

Thank you to Dr. Joan Francioni and Dr. Mingrui Zhang for advising and review.

7. REFERENCES

[1] Cleanflight, GitHub repository, Retrieved February 15, 2016, from Github: https://github.com/cleanflight/cleanflight.

[2] ESP8266 Arduino, GitHub repository, Retrieved February 11, 2016, from Github: https://github.com/esp8266/Arduino.

[3] *ESP8266 SDK Getting Started Guide*, Espressif, 2016.

[4] Liang, Oscar, *CleanFlight Setup Tuning Guide for Naze32 / CC3D*, Retrieved February 11, 2016, from Oscar Liang Blog: http://blog.oscarliang.net/cleanflight-naze32-setup.

[5] Liang, Oscar, *Naze32 on Mini Quad Setup Quadcopter*, Retrieved February 11, 2016, from Oscar Liang Blog: http://blog.oscarliang.net/naze32-on-mini-quad-fpv250.

[6] Tan, Wen, Jizhen Liu, Tongwen Chen, and Horacio J. Marquez, *Comparison of Some Well-known PID Tuning Formulas*, Retrieved February 11, 2016, from Science Direct: http://www.sciencedirect.com/science/article/pii/S009813540600

Distance Accuracies with Mobile Applications and Tracking Watches

Nathan Karlstad, Computer Science Department, Winona State University

Winona, MN 55987

NKarlstad12@winona.edu

ABSTRACT

The ubiquity of fitness-related mobile applications leads to consumer confusion and doubts in their choice. It can be frustrating for consumers if applications cannot distinguish flat and elevated terrain. The sample to experiment with includes 11 mobile applications and three tracking watches, which rely on the device or GPS to gather data. Findings show there is an issue in accurate distance recordings from at least half of the samples, and there is a significant problem in accurately recording distance in drastic changing altitudes.

1. INTRODUCTION

In the last 30 years, the lifestyles and food consumption of Americans have changed drastically, from children walking to school to riding the bus, to six snacks per day up from one [12]. Due to these changes, "more than one-third of adults and 17% of youth in the United States are obese" [16]. There is an unfortunate trend of poor diet and inactivity. The reasons vary from each individual, so in an effort to combat obesity, there has been a rise in various products and services related to weight loss and fitness. Since 2005, the number of health clubs increased by 7,630 to 34,460 total, and memberships to those clubs rose by 12.8 million to 54.1 million total [11]. For non-club members, there are independent options to motivate the individual and track their results such as mobile applications and peripherals.

With the ubiquity of smartphones, there has been a plethora of fitness-based mobile applications created on Google's application store, Google Play, alone [6]. Whether the user chooses to use their smartphone or a worn peripheral, the device records various data: number of steps, distance traveled, calorie outtake, sleep, and other related information. The data field, distance traveled, is interesting due to the various ways it could be calculated.

Expected ways the applications will calculate distance is by the number of recorded steps, the use of the smartphone's accelerometer, and GPS. It is unlikely any of the selected mobile applications will base their calculated distance on the number of steps; it would be too inaccurate. The use of an accelerometer will record a user's motion in a 3D grid [19], but given the unpredictable motions a person makes, the algorithms used will show a difference between applications in recorded distance. The use of GPS will be similar to the accelerometer, but it will be more accurate, within 95% [8]. However, like with the accelerometer, how the developers program the applications and devices to account for altitude will have an effect on the recorded distance. Some applications may only take into account latitude and longitude, which will assume the user is always on a flat plane.

For the average person, the exact distance may not be important, only that it is reasonably close, but for individuals committed to downloading applications related to fitness and paying for companion peripherals, the accuracy of the displayed distance may be important. It could prove frustrating if the device and/or application do not differentiate running on a treadmill and skiing down a mountain.

Because of the mentioned accuracy of GPS, it is expected the mobile applications' and tracking watches' recorded distance to be within the margin of error of five percent when compared to the measuring wheel's recorded distance. However, with the experiment involving both software and wearable hardware, it is expected the wearable devices will be 10% more accurate in recorded distance compared to the software applications on the smartphone.

2. METHODOLOGY

2.1 Materials and Software

There are two Fitbit watches: our own Charge [4] and Dr. Zhang's Surge [5]. The third watch is Dr. Cichanowski's Garmin [7]. The Fitbit watches will be worn on the left arm with the Surge closest to the wrist, and the Garmin watch will be worn on the right arm. The mobile applications include: Strava Running and Cycling GPS [20], Google Fit - Fitness Tracking [9], Walk with Map My Walk [14], Endomondo - Running & Walking [3], Pedometer & Fitness Tracker [18], Pedometer [21], GPS Odometer [10], Nike+ Running [15], Sports Tracker Running Cycling [2], Runtastic Running & Fitness [17], and WalkLogger pedometer [27]. These applications were chosen on developer name recognition (well-known and unknown developers) and number of downloads. Applications will run on the HTC One M8 smartphone using the Android operating system. The smartphone will be placed in the front-left pocket of the scientist's pants.

Proceedings of the 16th Winona Computer Science Undergraduate Research Seminar, April 27, 2016, Winona, MN, US.

An analog measuring wheel to record a distance in feet and inches to later be converted to kilometers, and a six foot leveling tool and a tape measure.

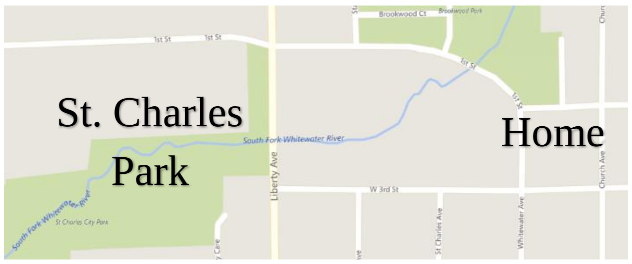
While the measuring wheel records the distance, the three watches will be worn, and the 11 applications will execute and record in the background on the smartphone. At the start of each trial, the measuring wheel will be zeroed and each application, if the distance is not displayed as zero, will be recorded to calculate the difference at the end of each trial.

2.2 Locations

The trail around Lake Winona (44°2'23.10"N, 91°39'2.37"W) will be used because it has a relatively flat elevation.

Lake Winona
Lake Park West
W Belleview St
Winona State University
Olmstead St
14th St
W Mill St
W Lake St
W 15th St
W 13th St
14th St
Johnson St
Huff St
Lake Winona
W Lake Blvd
14
Huff St
W Lake Blvd
E Lake Blvd
Garvin Heights Rd

A path from our home (43°58'33.76"N, 92° 3'48.91"W) to St. Charles' Park will be another location with its mix of flat and sloping terrain.



To test the accuracy in high altitudes, Holzinger Lodge (44° 2' 27.0085"N, 91° 39' 43.8324"W) will be the last location.

The locations will be tested three times to get an average. The distance measured by the measuring wheel will be targeting 8,100 feet for each location.

Uniquely, a hill with pavement will be used to predict what a device only calculating based on latitude and longitude will conclude. The hill is next to our home. It slopes upward/downward at a steady angle, representing a triangle. Using the equation *tan(θ) = h/d*, the height *h* can be found with the distance from where we stood and the bottom of the hill *d* and angle of the slope *θ*. The measuring wheel will be used to record the hypotenuse, and the angle will be calculated with a six foot leveling tool and a tape measure. The tools will form three separate smaller triangles along the road with the road being the hypotenuse. With one angle being 90˚, the remaining two can be calculated, for each triangle.

3. RESULTS AND ANALYSIS

With the use of the sloping road, a prediction of what applications and devices calculating distance without the inclusion of elevation would record is calculated. The equation,

$$\tan(\theta) = \frac{h}{d}$$

is used to calculate the height h of the triangle. The distance from where we stood to the bottom of the hill was recorded and averaged as 0.007 km for d, and the angle was measured and averaged as 48.64˚ for θ.

$$\tan(48.64°) = \frac{h}{0.007} km$$

$$h = 0.008\ km$$

With the hypotenuse and height, the predicted result, or third side, can be calculated with the equation,

$$a^2 + b^2 = c^2$$

The hypotenuse being c, and the height being a.

$$0.008^2 km + b^2 = 0.044^2 km$$

$$b = 0.043\ km$$

Table 1. Average recorded distance in km along the hypotenuse (the road)

Application and Device name	Average (Ascending)
Google Fit - Fitness Tracking	0.00
WalkLogger pedometer	0.00
Runtastic Running & Fitness	0.033
Pedometer	0.048
GPS Odometer	0.050
Garmin	0.053
Charge	0.053
Surge	0.053
Nike+ Running	0.060
Walk with Map My Walk	0.076
Sports Tracker Running Cycling	0.090
Endomondo - Running & Walking	0.097
Pedometer & Fitness Tracker	0.100
Strava Running and Cycling GPS	0.100

Table 1. shows the applications' and devices' recorded distance along the road in ascending order. Google Fit – Fitness Tracking and WalkLogger pedometer recorded a zero likely because both applications do not record past one decimal point; they are included but ignored for the sake of this discussion. As for the rest, the only application that recorded less than the measured road, 0.044 km, is Runtastic Running & Fitness, 0.033 km. This is 0.01 shorter than the predicted result, and the rest of the applications and devices recorded much higher values. It is possible that because we produced more movement climbing the hill than we would walking on a flat, straight line, the applications and tracking watches over calculated. However, this does not explain why a GPS-based application or device recorded incorrectly. The extremely short distance hindered Google Fit – Fitness Tracking and WalkLogger pedometer from recording, so the similar anomalies may have occurred for the applications that recorded high.

Figure 1. shows the data collected in alphabetical order. In each cluster, the left-most (purple) bar represents the data gathered and averaged for the particular application or device at Lake Winona. Eight applications and devices, Charge, Endomondo – Running & Walking, GPS Odometer, Nike+ Running, Pedometer & Fitness Tracker, Sports Tracker Running Cycling, Strava Running and Cycling GPS, and Walk with Map My Walk, recorded higher than the measurement gathered from the measuring wheel, 2.47 km.

The middle (orange) bar represents the averaged data recorded at St. Charles Park. Nine applications and devices, Charge, Endomondo – Running & Walking, Google Fit- Fitness Tracking, Nike+ Running, Pedometer & Fitness Tracker, Sports Tracker Running Cycling, Strava Running and Cycling GPS, Walk with Map My Walk, and WalkLogger pedometer, recorded above the measuring wheel, 2.47 km.

The right-most (green) bar represents the averaged data collected from Holzinger Lodge. Nine applications and devices, Charge, Endomondo – Running & Walking, Google Fit – Fitness Tracking, Nike+ Running, Pedometer, Pedometer & Fitness Tracker, Sports Tracker Running Cycling, Strava Running and Cycling GPS, Surge, and Walk with Map My Walk, measured higher than the 2.47 km from the measuring wheel.

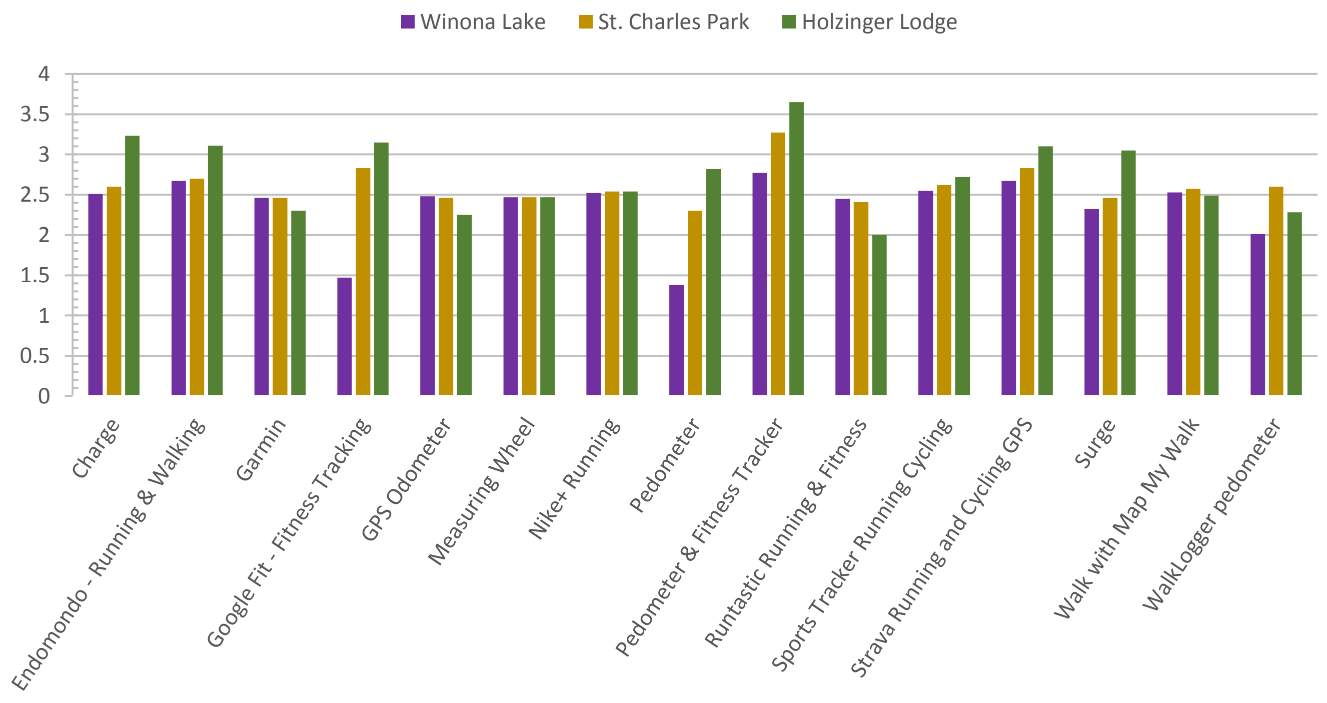


Figure 1. Average recorded distances for each location, Lake Winona, St. Charles Park, and Holzinger Lodge, in kilometers.

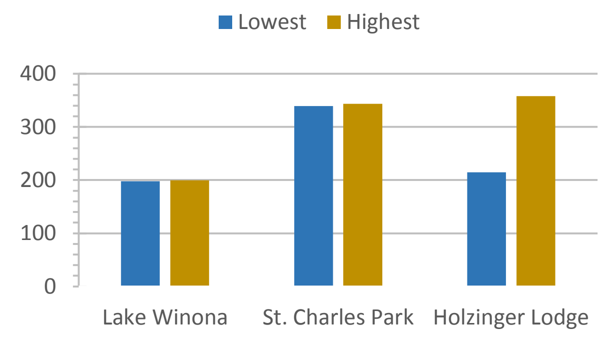


Figure 2. Lowest and highest recorded altitude from each location in meters.

According to Altitude.nu [1], which uses Google Maps, we measured the altitude of the three locations. Figure 2. displays the lowest and highest evaluated altitude for each location. As shown, the altitude increased by 139.64 m from Lake Winona to St. Charles Park. This may explain the trend for some of the applications and devices to record higher at St. Charles Park than Lake Winona.

However, when tested against the first hypothesis, seven applications and devices, Charge, Garmin, GPS Odometer, Nike+ Running, Runtastic Running & Fitness, Sports Tracker Running Cycling, and Walk with Map My Walk, are within the margin of error of five percent, supportive of the hypothesis, at Lake Winona. Eight applications and devices, Charge, Garmin, GPS Odometer, Nike+ Running, Runtastic Running & Fitness, Surge, Walk with Map My Walk, and WalkLogger pedometer, support the first hypothesis at St. Charles Park. The difference between the number of applications and devices within five percent of the measuring wheel at both locations is negligible. Figure 2. shows while there is a difference in height above sea level between Lake Winona and St. Charles Park, at each location, the difference between the lowest and highest elevation is not significant, meaning the paths were fairly consistent and relatively flat. The comparison between Lake Winona and St. Charles Park show there isn't enough reason to suggest a high altitude significantly affects distance measured by the applications and devices.

However, this experiment also focused on sloping terrain, changes in elevation throughout a location. Holzinger Lodge's lowest recorded altitude, from Figure 2., is 17.01 m higher than

Lake Winona's lowest, and its highest recorded altitude is 14.33 m higher than St. Charles Park's highest. It has a significant shift in elevation throughout its location. In Figure 1., only two, Nike+ Running and Walk with Map My Walk, are within five percent of the measuring wheel. It is a significant drop from at least 50% of the applications and devices supporting the hypothesis (seven out of 14) to 14.29%, or two out of 14, supporting the first hypothesis. Holzinger Lodge's lowest and highest altitude is not significantly lower or higher than Lake Winona or St. Charles Park, but the results displayed at Figure 1. show less applications calculating accurate distances at Holzinger Lodge compared to Lake Winona or St. Charles Park. These results show there is an issue, for these applications and devices, to record distance accurately on terrain that varies in altitude.

Table 2. Shows the standard deviation at each location and p-value

App/Device	Lake Winona	St. Charles Park	Holzinger Lodge	p-value
Charge	1	1	2	0.305
Endomondo – Running & Walking	1	1	2	0.129
Garmin	-1	-1	-1	0.357
Google Fit – Fitness Tracking	-3	2	2	0.982
GPS Odometer	1	-1	-1	0.424
Nike+ Running	-1	1	1	0.011
Pedometer	-3	-1	1	0.546
Pedometer & Fitness Tracker	1	3	3	0.096
Runtastic Running & Fitness	1	-1	-1	0.330
Sports Tracker Running Cycling	1	1	1	0.083
Strava Running and Cycling GPS	1	2	2	0.087
Surge	-1	-1	2	0.595
Walk with Map My Walk	1	1	1	0.122
WalkLogger pedometer	-1	1	-1	0.416

The standard deviation was calculated for each location, as shown in Table 2. The standard equation used had to be modified to replace the overall mean with the mean of the measuring wheel. For Lake Winona, the standard deviation was 0.4478. For St. Charles Park, 0.2850. For Holzinger Lodge, 0.5616.

Table 2. also shows the p-value from a conducted T-test. For the main hypothesis, the first one, the alpha is 0.05. The mobile application, Nike+ Running, was the only one to pass with a 0.011, meaning its data is significant. However, since only three samples were collected at only three locations for each application and device, there is not enough data to confidently state if the significance of the data holds. Pedometer & Fitness Tracker, Sports Tracker Running Cycling, and Strava Running and Cycling GPS could also be argued to have significant data given their values are under 0.100. Google Fit – Fitness Tracking likely has a 0.982 due to its wide range in given data, which reinforces the idea there was not enough data collected.

For the second hypothesis, at Lake Winona, the Charge recorded 10% better than four applications: Google Fit – Fitness Tracking, Pedometer, Pedometer & Fitness Tracker, and WalkLogger pedometer. The Surge did better than three applications: Google Fit – Fitness Tracking, Pedometer, and WalkLogger pedometer. The Garmin produced more accurate results than four mobile applications: Google Fit – Fitness Tracking, Pedometer, Pedometer & Fitness Tracker, and WalkLogger pedometer.

At St. Charles Park, Charge, Surge, and Garmin recorded 10% more accurately than three applications: Google Fit – Fitness Tracking, Pedometer & Fitness Tracker, and Strava Running and Cycling GPS.

At Holzinger Lodge, Charge and Surge produced more accurate recordings by 10% than Strava Running and Cycling GPS. Garmin did better than five mobile applications: Endomondo – Running & Walking, Google Fit – Fitness Tracking, Pedometer & Fitness Tracker, Runtastic Running & Fitness, and Strava Running and Cycling GPS. The number rises to seven if Charge and Surge are included.

The tracking watches did not perform significantly better in producing accurate distance recordings, dismissing the second hypothesis. The watches tend to record more accurately on level terrain, not on hiking trails like Holzinger Lodge, in which they perform similar to mobile applications.

4. CONCLUSIONS

Figure 1. revealed an interesting trend: the orange, middle, bars representing St. Charles Park, appeared to be recording higher results than Lake Winona. This made us question whether the altitude played a significant role in recording accurate distance. Holzinger Lodge is a near-perfect location to test the recordings due to its significant change in altitude. However, it is a forest, which is in contrast to the near-woodless locations of Lake Winona and St. Charles Park. It is unclear if the dense woods interfered with some of the applications' recordings. In our experience, GPS Odometer warned of a weak GPS signal from the base of Holzinger Lodge, and Nike+ Running began saying, "Pause workout," halfway through the path. As discussed in Table 1., human arm movement uphill is more exaggerated to keep balance and more quick downhill. Holzinger Lodge consisted of many steep hills, which had to be climbed up and down. This may have resulted in the high values generated by the Charge and Surge. Mobile applications relying on the smartphone's accelerometer would have also likely suffered from human error, due to similar reasons discussed about arm movement, but since the smartphone was placed in the left front pocket, the error likely occurred from exaggerated leg movement. For example, going down a steep hill results in quick or short steps to not lose balance.

While this experiment provided a decent sample of applications and popular tracking watches, a larger sample size is always beneficial. If this test were to be conducted again, more locations would also be chosen along with more tests at each location. In this study, there is a problem with applications and devices recording in drastic shifts in elevation. More locations, specifically hiking locations like Holzinger Lodge, would be best to support this conclusion.

5. ACKNOWLEDGEMENTS

We would like to thank Dr. Zhang and Dr. Cichanowski for lending their tracking watches. We would also like to thank Dr. Zhang and Dr. Francioni for providing helpful feedback in data collection, writing, poster design, and presentation. They hosted the course CS 495, and each student also provided meaningful feedback.

6. REFERENCES

[1] Altitude.nu. Altitude.nu - Find the Elevation of Any Place. N.p., n.d. Web. 19 Apr. 2016. <http://altitude.nu/>.

[2] Amer Sports Digital Oy. "Sports Tracker Running Cycling." Sports Tracker Running Cycling - Android Apps on Google Play. Google Play, 8 Mar. 2016. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.stt.android>.

[3] Endomondo.com. "Endomondo - Running & Walking." Endomondo - Running & Walking - Android Apps on Google Play. Google Play, 18 Feb. 2016. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.endomondo.android&hl=en>.

[4] Fitbit. "Energize Your Day." Fitbit Charge™ Wireless Activity + Sleep Wristband. Fitbit, n.d. Web. 10 Mar. 2016. <https://www.fitbit.com/charge>.

[5] Fitbit. "Train Smarter. Go Farther." Fitbit Surge™ Fitness Super Watch. Fitbit, n.d. Web. 10 Mar. 2016. <http://www.fitbit.com/surge>.

[6] "Fitness+tracker - Android Apps on Google Play." Google Play. Google, n.d. Web. 29 Jan. 2016. <https://play.google.com/store/search?q=fitness%2Btracker&c=apps&docType=1&sp=CAFiEQoPZml0bmVzcyB0cmFja2VyegUYAMABAooBAggB%3AS%3AANO1ljIdNaE>.

[7] Garmin. "Garmin | United States | Home." Garmin. Garmin, n.d. Web. 10 Mar. 2016. <http://www.garmin.com/en-US>.

[8] "Global Positioning System Standard Positioning Service Performance Standard." (2008): 11. GPS.gov. Sept. 2008. PDF. 11 Feb. 2016. <http://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf>.

[9] Google, Inc. "Google Fit - Fitness Tracking." Google Fit - Fitness Tracking - Android Apps on Google Play. Google Play, 8 Jan. 2016. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.google.android.apps.fitness>.

[10] GPS Tools. "GPS Odometer." GPS Odometer - Android Apps on Google Play. Google Play, 10 Apr. 2015. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.gpstools.gpsodometer>.

[11] "IHRSA - About the Industry." IHRSA. N.p., 30 June 2015. Web. 11 Feb. 2016. <http://www.ihrsa.org/about-the-industry/>.

[12] "Let's Move." Let's Move. N.p., n.d. Web. 11 Feb. 2016. <http://www.letsmove.gov/learn-facts/epidemic-childhood-obesity>.

[13] Liu, Guangwen, Masayuki Iwai, Yoshito Tobe, Dunstan Matekenya, Khan Hossain, Masaki Ito, and Kaoru Sezaki. UbiComp '14 Adjunct Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication. Ubiquitous Computing, New York. New York: ACM, 2014. 459-68. ACM Digital Libary. Web. 22 Jan. 2016.

[14] MapMyFitness, Inc. "Walk with Map My Walk." Walk with Map My Walk - Android Apps on Google Play. Google Play, 1 Mar. 2016. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.mapmywalk.android2&hl=en>.

[15] Nike, Inc. "Nike+ Running." Nike+ Running - Android Apps on Google Play. Google Play, 2 Feb. 2016. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.nike.plusgps>.

[16] Ogden CL, Carroll MD, Kit BK, Flegal KM. Prevalence of Childhood and Adult Obesity in the United States, 2011-2012. JAMA.2014;311(8):806-814. doi:10.1001/jama.2014.732.

[17] Runtastic. "Runtastic Running & Fitness." Runtastic Running & Fitness - Android Apps on Google Play. Google Play, 5 Feb. 2016. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.runtastic.android>.

[18] SenseMe. "Pedometer & Fitness Tracker." Pedometer & Fitness Tracker - Android Apps on Google Play. Google Play, 10 June 2015. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.wearablelab.fitnessmate>.

[19] Smith, Dave. "How Does An Accelerometer Work In A Smartphone? Bill Hammack, The Engineer Guy, Explains [FULL TEXT]." International Business Times. N.p., 23 May 2012. Web. 11 Feb. 2016. <http://www.ibtimes.com/how-does-accelerometer-work-smartphone-bill-hammack-engineer-guy-explains-full-text-699762>.

[20] Strava, Inc. "Strava Running and Cycling GPS." Strava Running and Cycling GPS - Android Apps on Google Play. Google Play, 8 Mar. 2016. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.strava>.

[21] Tayutau. "Pedometer." Pedometer - Android Apps on Google Play. Google Play, 7 Sept. 2015. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.tayu.tau.pedometer&hl=en>.

[22] Walklogger. "WalkLogger Pedometer." WalkLogger Pedometer - Android Apps on Google Play. Google Play, 6 Jan. 2015. Web. 10 Mar. 2016. <https://play.google.com/store/apps/details?id=com.walklogger.pedometer>.

Java Implementation of Cox Proportional Hazards Model

Nathan Martin
Winona State University
178 East 6th St Winona, MN 55987
763-370-5275
NMartin11@winona.edu

Abstract

The project is to redesign the web-based software for predicting lung cancer treatment outcomes. The current application was designed eight years ago and has been used at Mayo Clinic since. It was done by integrating R statistical package into Java environment. It introduced unnecessary complexity due to the need of the Cox Proportional Hazard model in R. To simplify the design, we developed a Java version of Cox Proportional Hazard model to compute patient survival rate. We both both ArrayList and HashMap of Java implementation for information exchange between web-based interfaces and the Java implementations. Our software tool calculates the survival rate using patient histological information and chosen treatments, and then presents the survival curves on web-based interfaces. In the redesign of the software tool, we have improved the user interfaces to allow a healthcare provider compare patient's survival rates between different treatments.

1. Introduction

The Cox proportional hazards model or simply known as the Cox model is a statistical technique for exploring the relationship between the survival of a patient and several explanatory variables [1]. This model is the most common tool for studying the dependency of survival time on predictor variables [2]. For example, this model can evaluate the current status of a patient with lung cancer and predict the life expectancy based on predictor variables. It can also provide an estimate of the treatment effect on survival after adjustment for other explanatory variables [1]. The hazard function represented as S(t|x), is the probability that a person will experience an event within a small time interval [1].

$$S(t|x) = \exp[-h_0(t)xe^{\beta x}] \qquad \text{Equation. 1}$$

Shown in Equation 1, $h_0(t)$ is the baseline which corresponds with the probability of dying when all given explanatory variables of x are at default values. When used properly we can express the hazard or risk (S(t)) of dying at time t [1]. In short, the Cox model is used to analyze survival data, which allows us to isolate the effects of treatment from the effects of other variables. Although the Cox model is commonly used for survival analysis, essentially the same methods are employed in a variety of disciplines under various rubrics such as "event-history analysis" in sociology, and "failure-time analysis" in engineering [2].

There are a few software tools that can effectively use the Cox model, they include R statistical language, SAS, Minitab, JMP, and STATA. None of them are in Java. The goal of this project will be creating a generic Java function to approximate cox model, and use it in different survival models such as Background model of non-small cell lung cancer (NSCLC), Treatment model of non-small cell lung cancer, Quality of life (QoL) model, extensive stage small cell lung cancer model, and limited stage small cell lung cancer model. Results from this application need to approximate to that of the results from R.

There is limited availability of the Cox PH model implemented in Java. A package that can be downloaded was developed by Campagne Laboratory at the institute for computational biomedicine of Weill Cornel Medical College, but isn't official to Java [5]. Also, there is another project called JStats started by a small group [4]. The problem with the JStats package is that on their website you are presented with a warning that the package is incomplete, and almost certainly neither usable nor functional [4].

For this project it is to be expected that Java can handle such models and output accurate results but will be limited to only the Cox PH model. I hypothesize that a carefully designed Java implementation of Cox model will compute survival rate within 5% of estimated results by R package. Because of the focus on the Cox model, this project will be limited to just that and will not perform other statistical analysis functions for the time being.

2. Methodology

Java is the programming language being used to create a function of Cox proportional hazard model. Our Java implementation will allow a user to pick a desired model and convert the data into a hash map which is a data structure stores both key and value. In this case each coefficient or time interval will be the key paired to its corresponding values.

Table 1. CSV file with coefficient and there values

Coefficient	Baseline
agedx	0.0256
gender	0.232644
stageib	0.187212
stageiia	0.297941
stageiib	0.549594

Looking at table 1, the first column is the key and the second column would be the value of the key. When we want to access a coefficient all is needed is to search for the name of the coefficient and value is returned.

Math.exp (-(1-baseline)*Math.exp (CoefficientSum))

In Java implementation, the CoefficientSum variable corresponds to the βX in Equation 1 which is the sum of all coefficient to be used in the equation. For example, if we used agedx and gender from table 1 the CoefficientSum will be equal to .258244. – (1 – baseline) is equivalent to $-h_0(t)$ in Equation 1 and its value is provided by the specified model we decide to use. Math.exp is Java's method of taking the exponent of the value in the parenthesis.

2.1 R Statistical Program

R is the statistical computing program that is used to compare the Java implementation results to determine the accuracy of the survival rate results. Since the focus of project is Java implementation, Dr. Mingrui Zhang of the Computer Science Department from Winona State University provides the data that will be used to compare and determine the validity of the Java implementation. This data includes results from NSCLC background, NSCLC treatment, post-surgery recurrence, quality of life, extensive small cell, limited small Cell.

Table 2. ***Sample data from R of Survival Rate for first 10 months with Non Small Cell Type Model***

Month	Baseline
0	0.99997265
1	0.99856295
2	0.99673442
3	0.99470675
4	0.99289737
5	0.99129227
6	0.98959672
7	0.98790793
8	0.98616069
9	0.98434493
10	0.9826557

In Table 2, the coefficients or predictor variables being used to get the survival rate is age being 50 and stage being stageiiib. Stageiiib is the stage of lung cancer the patient is at. Numbers in the first column are the months. The second column contains the base survival rate corresponding to each month. You can see that as each month passes the survival rate or probability of survival decreases. Looking back at Equation 1, $h_0(t)$ is the base at a given month. For example, at month 0 $h_0(t)$ is set to the corresponding survival rate.

2.2 Significance Testing

IBM SPSS Statistics 22 is used in computing the ANOVA and t-tests, it provides all needed data to determine the validity of the Java implementation. Data such as means, standard deviation and variance of each data set are computed using this software and compared. In order to test the application against the one provided by R there will be a set of different data inputs each including coefficients and baseline values that are needed for the application to work. Each application (Java and R) takes in the different inputs and produce the results. Results produced by the Java application are compared to the ones produced by R. When comparing the two results, the mean values and standard deviations of the differences between two curves need to be computed and analyzed. A t-test can then be performed to see if the difference is significant.

2.3 Software Development Model: AGILE

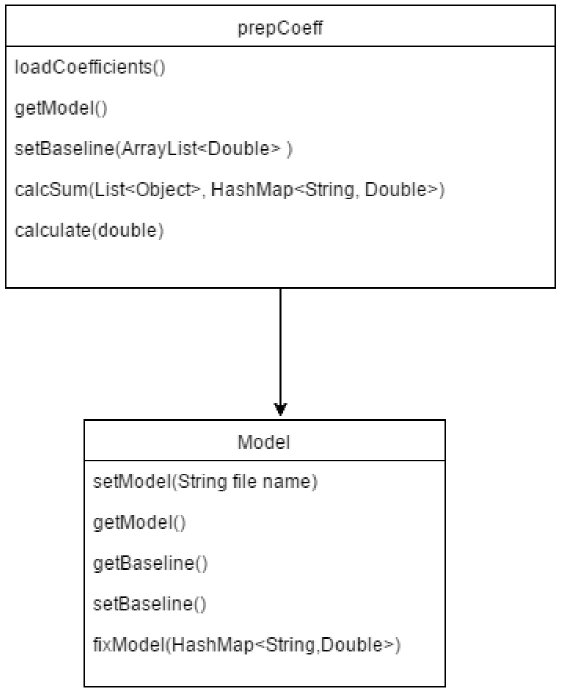

prepCoeff
loadCoefficients()
getModel()
setBaseline(ArrayList<Double>)
calcSum(List<Object>, HashMap<String, Double>)
calculate(double)
Model
setModel(String file name)
getModel()
getBaseline()
setBaseline()
fixModel(HashMap<String,Double>)

Figure 1. Class Diagram of the Cox Model program in Java

The software development model that we have followed is the Agile model. This process allows for small incremental improvements with minimal planning. Each iteration are within short time frames and involve all members working on planning, requirement analysis, design, coding, and testing. Once an iteration is done, it is then reviewed and the next round of development precedes. This allows for a project to take in account of unpredictability and adapt to needed changes. Figure 1 shows the methods used in the Java class. In the prepCoeff class, the load coefficient is used for the user to enter the coefficients that need to use. Get model method returns the model being used and the set baseline method gets the baseline values of each month in the model and puts it in an array list. Most importantly, the calcsum method adds the values of the coefficients that the user specified and gets it ready to pass the value to the calculate method. The calculate method is where the Cox Proportional hazard model is used to compute the survival rate. As for the Model class, it puts the model being used into a hashmap and set the baseline value for that specified model. This is not what the final class will look like but rather the bare essentials.

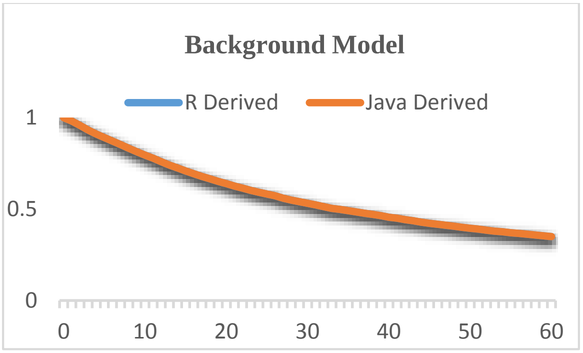


Figure: 2. *Survival curve produced by Java function compared to results from R*

In Figure 2 the line graph there is a descending curve which shows over time the probability of survival given the predictor variables which are the age and stage. In the graph the x-axis represents the time which is in months. In this case the x-axis goes to sixty months which is equivalent to 5 years. The y-axis is the survival rate which starts near one or one hundred percent. The line labeled R derived is the survival data set that was produced by R package which is the same set previously mentioned in Table 2. Java derived is the line produced by the current function that was created in Java.

2.4 T-Test

T-test is used to compare two different data sets which, in this experiment, the results produced from the Java implementation and the results from R. To conduct a t-test, the means, standard deviation, variance are calculated for each set of data. Using these the means, standard deviation, variance, we calculate the t value. This can be done by using Equation 3.

$$t = \frac{mean_1 - mean_2}{s} \qquad \text{Equation 2}$$

In Equation 2, $mean_1$ is the mean of the first set of data by the calculating the summation of the set and divided by the number of values in the set. $Mean_1$ is the results from the R package. $Mean_2$ is the mean of the second set of data which is the result of the Java function. We then get the difference of $mean_1$ from $mean_2$ and divide it by the Standard Deviation. To get the standard deviation, use Equation 3.

$$s = \sqrt{\frac{\Sigma(x-\bar{x})^2}{n-1}} \qquad \text{Equation 3}$$

The x in Equation 3 is the value of the variable in the set, and $\bar{x}$ is the mean of the data set x. This means we take the value of each variable in the set subtracted by the mean of the set, then take the summation of them and square the result. N is the total number of values in the data set. Finally we take the square root of the result which will give us the standard deviation.

In addition the cut off of significance is 0.05. Finally, we compare the cut off level and the value of t to see if results from the Java implementation is significant or not.

3. Results and Analysis

Table 3: Shows the mean, Standard Deviation, and Standard Error of Mean

Test data	Mean	Std. D	Std. Error Mean
R Background	0.5809	0.18732	0.02398
Java Background	0.5809	0.18732	0.02398
R limited	0.6789	0.17279	0.02373
Java Limited	0.6822	0.16921	0.02324
R Extensive	0.6027	0.21722	0.03352
Java Extensive	0.6119	0.21138	0.03262
R QOL	0.7023	0.1797	0.0232
Java QOL	0.7114	0.18083	0.02335
R Recurrence	0.3369	0.23524	0.03976
Java Recurrence	0.34048	0.23169	0.03916

In the above table, we compared the results derived from R and results derived from the Java functions for five different clinical models. They are Background, Limited, Extensive, Quality of Life, and Post-surgery Recurrence models. Each model has their own set of coefficients with a corresponding value along with their own baseline data. Some models have a few gaps in data which will be accounted for by filling in missing data. We have calculated the Mean, Standard deviation, and Standard Error Mean of the differences between results. Looking at the R Background row, the mean was calculated by taking the results given by R, adding up all the values and dividing by the number of values. This gives us the average survival rate of the data set. Looking at the Std. D or Standard Deviation which tells us the average variance in the data that the difference in each set for a model is minimal. By examining the results, it is clear that the Java function performs very similar to that of R with results almost identical and well within the confidence interval shown in table 3.

Table 4 is extension of the results of Table 3 by performing a paired sample t-test. A t-test is used to compare two set of means from two samples that are correlated. Taking the first row as an example. The t-test for Background compared the means, standard deviation, and significance of the results from R and Java when using the Background model. For the columns Mean and Std. D the value is telling us the difference between the R and Java functions results. This means that there is no difference between R and Java when using the background model. Because the mean and Std. D are zero in the first column the significance is unambiguous. As for the models limited, extensive, QOL, and recurrence since there is some variations between R and Java for each model we have to look at the value of the Sig column. If the value in this column is below .05 it means that the result is significant and assume that they are not likely due to change.

Table 4. Shows the results of T-test between R and Java

T-test	Mean	Std.D	Sig
Background	0	0	0
Limited	-0.00329	0.00677	0.001
Extensive	-0.00473	0.01251	0.019
QOL	-0.00902	0.00537	0
Recurrence	-0.00386	0.00981	0.014

To further iterate the significance of our tests we can look at Figure 3. This graph includes two curves that represent the results for the Quality of Life model. The orange line represents the results calculated by the Java function and the blue line represents the results from R.

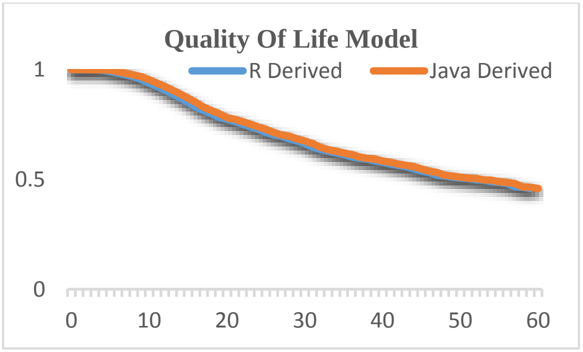


Figure 3. Graph showing results from R and Java implementations

The Java derived results follow the same curve shape as the R derived results. Results from Table 4 tell us that the difference between R and Java for the quality of life model are minimal. Figure 3 also shows us that there is minimal difference between the two functions.

Figure 4 shows results from R and Java for the post recurrence model. Our Java implementation follows the curve for R. The difference between Figure 3 and Figure 4 is that our Java function accounts for gaps of data in the model and completes it. This is why the Java curve extends the full 60 months as compared to R stopping at 35 months.

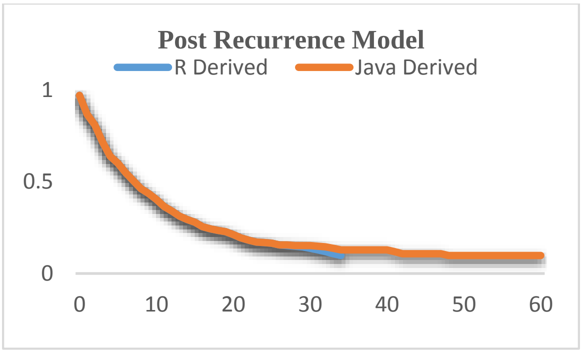


Figure 4. Graph showing results from R and Java for post recurrence model

4. Importance of Project

The completion of this project simplifies and reduces the complexity of the web based software for predicting lung cancer treatment outcomes. It replaces the need for the R language with the Java implementation of the Cox Proportional Hazard Model. This also enables the project to be more light weight, easier to maintain, and deployed on web servers for future iterations of the software. Overall, the project as a whole brings an important tool for lung cancer patients to compare their treatment options and make a more informed decision that will have a life lasting effect.

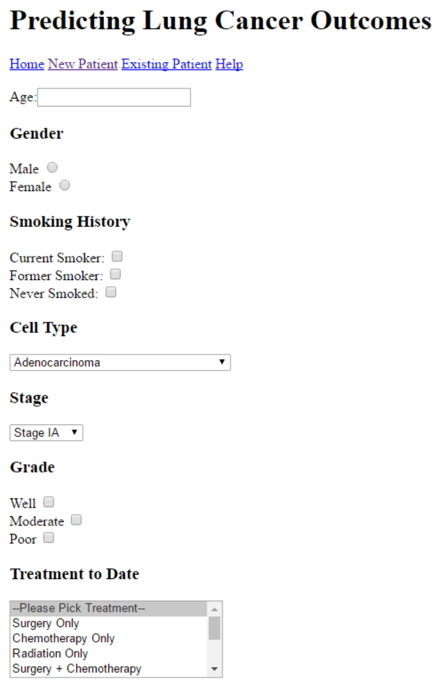


Figure 5. User interface for lung cancer prediction tool

Because of the simplifications of the software future development can be spent on putting more resources on the user interface to make it easier for doctors and patients to fill out and receive the information they need. Currently the user interface is kept simple and has a few forms for the user to fill out. The first page is used to gather the needed information to determine what model should be used. Depending on what is chosen you will get sent to a different page which will display a curve or ask for more data. Our UI also allows for users to use it on multiple different devices because of its responsive design.

5. Conclusion

With the current results, it is concluded that the Java function outputs survival rate with the accuracy of the function used in R. Further development can make the function more generic, allowing it to work properly with more models and allow us to perform more extensive testing assuring the use of different coefficients will still produce accurate results. However, currently there are some limitations to the current program. For a user to use the function we have created they need to know the locations of their files and have some knowledge as to what coefficients are needed for it. If they do not know the proper guidelines for the model they want to use, they will either get an error message saying "invalid data" or they will get inaccurate results.

6. Acknowledgments

I would like to thank Dr. Mingrui Zhang for including me and helping me throughout this project. He conducted code reviews and provided me with helpful articles and suggestions. I would also like to thank the Winona State Computer Science Department for this opportunity.

References

[1] Walters, Stephen John. What Is a Cox Model? 2nd ed. England: Hayward Medical Communications, 2009. What Is…? What is a Cox Model, May 2009. Web. 21 Jan. 2016

[2] Fox, Brown and Sanford Weisberg. "Cox Proportional Hazards Regression. " Encyclopedia of Public Health 2 (2011): 1-20. 23 Feb. 2011. Web. 21 Jan 2016.

[3] Spatz, Chris, Exploring Statistics: Tales of Distributions. 11th ed. Conway: Outcrop, 2016. Print.

[4] Edelstein, David, and Justin Scheiber. "The Fnordistan Deartment of Software Engineering." JStats, N.p. , n.d. Web. 28 Jan 2016.

[5] Campagne, Fabien. "Campagne Laboratory." BDVal – Campagnelab, Well Medical College of Cornell University, 4 Aug. 2010. Web. 10 Feb. 2016

[6] Urbanek, Simon. "Low-Level R to Java Interface." (2016): 1-45. Cran R Project. Web.

Performance Analysis of ArrayList and HashMap

Asiqur Rahman
419 ½ Olmstead St
Winona, MN 55987
ARahman12@winona.edu

ABSTRACT

A Java Collection Class is a data Structure, which is used for processing data. In general, finding a good balance between memory utilization and time efficiency is quite challenging. ArrayList and HashMap are two commonly used Java Collection Classes. They store and process the data differently. While ArrayList implements List interface and extends AbstractList class, HashMap implements Map interface and extends AbstractMap class. In this paper, a performance analysis is conducted on ArrayList and HashMap. If there are more than 0.1 million records, our experiment shows that ArrayList uses approximately 20-40% less memory than HashMap and provides similar performance up to 0.6 million records for randomly accessing to the data.

Keywords

ArrayList, HashMap, Memory Usage, Overhead Time, Access Time.

1. INTRODUCTION

A Collection class is a single object representing a group of objects, known as its element. These Collection class can be referred to as containers for other object [5]. It can perform the operations including searching, sorting, insertion, manipulation, deletion etc. Above all, Java Collection classes reduce the programming effort by providing useful data structures and algorithms.

Applications often have large runtime memory requirements. In some cases, this large memory footprint helps to accomplish an important functional, performance, or engineering requirement. However, finding a good balance between memory consumption and time efficiency is quite challenging. To do so, the development team must distinguish effective from excessive use of Memory and Time [7].

Due to the above facts the developers often get confused about finding a well-balanced data structure that they should use within their project. HashMap and ArrayList are two frequently used collection classes in Java. An ArrayList implements the List interface, which is an ordered collection of objects that we access using an index, much like an array. A HashMap implements the Map interfaces, which is an object that maps keys to values. A map cannot contain duplicate keys and each key can map to at most one value [4].

Proceedings of the 16th Winona Computer Science Undergraduate Research Seminar, April 27, 2016, Winona, MN, US.

1.1. ArrayList

ArrayList is one of the popular java collection classes. It implements List interface and extends AbstractList class (Figure 1). It stores one object that holds a reference to the value. The Java Platform SE 6 API documentation describes ArrayList as:

"An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list. Unlike sets, lists typically allow duplicate elements [2]."

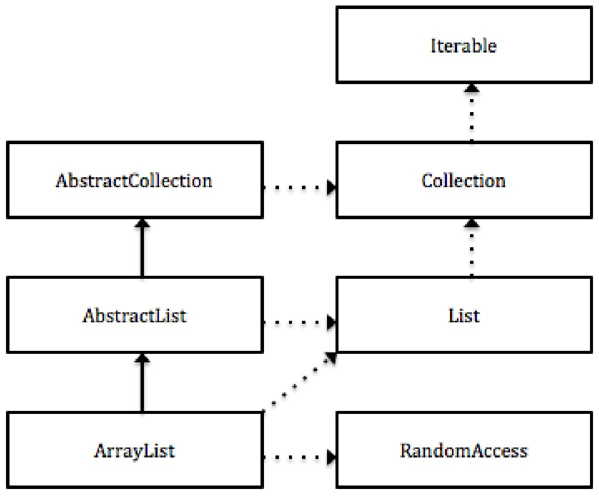


Figure 1. Internal design of ArrayList (Collected from Eclipse)

ArrayList stores the data internally in an array. When an ArrayList is being initialized, an array of size 10 is created by default when no parameter is passed and all elements are added to this array. But the ArrayList size can also be passed to any number as a parameter while initializing the ArrayList. When adding a new element, if the array is full, a new array is created with double the initial size, in order to accommodate all the elements in the ArrayList. An empty ArrayList uses 88 bytes of memory. For a 10,000-entry ArrayList, the overhead of ArrayList objects is approximately 40K [2].

1.2. HashMap

HashMap is another popular java collection class. It implements Map interface and extends AbstractMap class. It stores two objects, one holds a reference to the value and another object holds a reference to the key that make a huge disparity in terms of memory consumption. HashMap works on the principle of Hashing. To understand Hashing, we should understand the three terms, i.e. Hash Function, Hash Value and Bucket [3].

- Hash Function: Hash Function is the hashCode () function, which returns an integer value.
- Hash Value: Hash Value is the integer value that is returned by the hashCode ().
- Bucket: A bucket is used to store key value pairs. A bucket can have multiple key-value pairs. In HashMap, bucket used simple linked list to store objects.

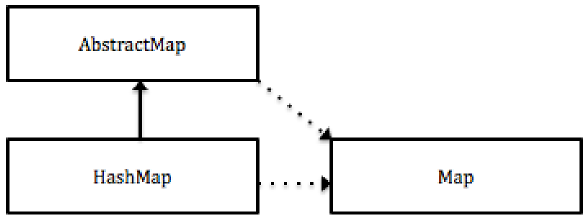


Figure 2. Internal design of HashMap. (Collected from Eclipse)

When we use put () method to store (Key, Value) pair, HashMap implementation calls hashCode on Key object to calculate a hash that is used to find a bucket where Entry object will be stored. When get () method is used to retrieve value, again key object is used to calculate a hash which is used then to find a bucket where that particular key is stored.

When a HashMap is created, the default capacity of HashMap object, which is an array of HashMap$Entry objects, is 16 entries. This gives a HashMap a size of 128 bytes when it is completely empty [2]. Any key/value pairs inserted into the HashMap are wrapped by a HashMap$Entry object, which itself has some overhead. The total overhead of a HashMap consists of the HashMap object, a HashMap$Entry array entry, and a HashMap$Entry object for each entry, which can be expressed by the following formula [2]:

$$H_O = H_{Obj} + A_O + (n * (H_{EAE} + H_{EObj}))$$

Where H_o is the total overhead, H_{Obj} is the HashMap object, A_o is the Array object overhead, n is the number of entries, H_{EAE} is the HashMap$Entry array entry, H_{EObj} is the HashMap$Entry object.

For a 10,000-entry HashMap, the overhead of just the HashMap, HashMap$Entry array, and HashMap$Entry objects is approximately 360K. This is before the size of the keys and values being stored are taken into account [2].

Our Hypothesis was ArrayList uses approximately 20-40% less memory spaces than HashMap with similar access time if n>=0.1 million where n= number of records, keys are integer and values are string.

In this paper, we conducted a performance analysis on ArrayList and HashMap by utilizing the Java Garbage Collection (JGC) and Java Built-in function nanoTime ().

2. SOFTWARE TOOLS

Java Garbage Collection (JGC) is an automatic memory management feature in Java Memory Management that runs within the JVM (Java Virtual Machine). It allows creating new objects without worrying explicitly about memory allocation and de-allocation, because the Garbage Collector automatically reclaims memory for reuse [8]. As application runs, new object gets allocated in the heap memory. And when the object is no longer referred any more in the program, JGC recollects those memories that were allocated for the uncalled object. In this project JGC was used after storing the data to the data structures to calculate the memory usage in the memory usage experiment.

Java System class contains several useful class fields and methods. nanoTime () is one of them. It returns the current time in nanoseconds (there are $1*10^9$ nanoseconds in a second). It was used during the overhead and access time experiments to calculate the elapsed time by getting the starting time and the ending time of the operations.

The experiments were conducted on the Operating System, OS X Yosemite, version: 10.10.5, Processor: 1.4 GHz Intel Core i5, Memory: 4 GB 1600 MHz DDR3, Graphics: Intel HD Graphics 5000 1536 MB, and the Eclipse IDE, version: 2.1.2.v20160212-1500, Build id: M20160212-1500

3. PERFORMANCE ANALYSIS

3.1 Approach

As part of the project, I did some comparison analysis. The objective was to find the memory usage, memory consumed by each data structures; overhead time, the amount of CPU time it needs to store the data; and access time, the amount of CPU time it needs to access the data for ArrayList and HashMap. Then I analyze their memory utilization, overhead time, and access time. In this experiments various numbers of datasets were used. Those datasets were collected from SCOWL (Spell Checker Oriented Word List) and Friends by Kevin Atkinson, PhD Graduate at University of Utah. Scowl is a collection of word lists split up in various sizes and other categories [1]. In this project thirteen different sizes of datasets were used, each of them were collected form SCOWL. For each dataset all the data are stored in a file in an Alphabetical order. The datasets contain en_US dictionary words ranging from 1 to 2000 thousands of words. Since HashMap stores two objects (key-value) and ArrayList stores one object as previously mentioned (section 1.1 & 1.2), we experimented the ArrayList using both single and double objects and compared them with the HashMap. We used the following algorithms to compute the memory usage, overhead time and access time for each dataset:

ALGORITHM 1: Memory Usage

```
1. Initialize data structures, ArrayList/HashMap
2. Read the data from the file
3. Add /Put data into the ArrayList / HashMap
4. Repeats step 3 & 4 until the file has element
5. Get the available memory and free memory
6. Finally, compute the used memory by subtracting the free
   memory from the available memory.
```

ALGORITHM 2: Overhead Time

```
1. Initialize data structures, ArrayList/HashMap
2. Get the Start time from the system
3. Read the data from the file
4. Add /Put data into the ArrayList / HashMap
5. Repeats step 3 & 4 until the file has element
6. Get the End time from the system
7. Finally, compute overhead time by subtracting the end
   time from the start time.
```

ALGORITHM 3: Access Time (last index/key)

```
1. Declare a Boolean flag exist.
2. Initialize data structures, ArrayList/HashMap
3. Read the data from the file
4. Add /Put data into the ArrayList / HashMap
5. Repeats step 3 & 4 until the file has element
6. Get the Start time from the system
7. Set exist $\leftarrow$ contain/containKey (Key/Value). The
   operations, contain/containKey will return Boolean
   values depending on whether the key/value exists in the
   list/map.
8. If exist is true, Get the End time from the system and
   compute access time by subtracting the end time from the
   start time. Otherwise, print out "key is not found".
```

ALGORITHM 4: Access Time (random)

```
1. Initialize data structures, ArrayList/HashMap
2. Read the data from the file
3. Add /Put data into the ArrayList / HashMap
4. Repeats step 3 & 4 until the file has element
5. Get the Start time from the system
6. For each key in keys (list)/key.Set () (map); Print out the
   key
7. Get the End time from the system
8. Finally, compute access time by subtracting the end time
   from the start time.
```

In this project the java built-in function getRuntime () were used to get the runtime instance that allows getting the available heap memory and free memory by running the garbage collector. The java built-in runtime instance functions, totalMemory () & freeMemory (), were used to get the runtime available memory and free memory respectively. For the access time the function nanoTime () were used to calculate the response time of these data structures. Since time measurement commonly referred to as wall (clock) can vary widely depending on the speed of the hardware it is running on, the efficiency language of the language translator and operating system, and the number of other process the platform is executing [6], we used the same eclipse IDE and hardware system for this experiments.

3.2 Results and Analysis

3.2.1. Memory Usage

In these experiments we found the significant differences when the numbers of records are exceeded 0.1 million. The result we got from the experiments are given below:

Table 1: Memory Usages (in kilobytes) for various numbers of records (in thousands).

Records	ArrayList			HashMap	
	I	II	III	I	II
1	2043	2043	2043	2043	2043
25	13113	13624	13701	14645	14643
50	9865	12141	12240	14015	14027
75	6574	10051	9950	12851	13122
100	17845	22227	22400	13839	14084
200	18546	28875	28812	36795	36553
300	34948	37963	37653	47205	47687
400	35261	57284	56988	60137	60641
600	52997	73136	73639	78765	81936
800	71958	98406	98845	116550	84825
1200	96746	140075	139860	170274	96410
1600	137133	187188	187225	233977	108737
2000	155827	206838	207463	277430	121606

Note that on the above table, column, ArrayList I is the measurement of single object. Columns, ArrayList II & HashMap I, are the measurements of key-value pair, where keys are integer and values are string. Columns, ArrayList III & HashMap II, are another key-value pair measurements where keys are string and values are integer.

Based on the above data four comparison graphs were plotted. One graph is the comparison between ArrayList (single object) and HashMap using string as the key and integer as the value. The next graph is the comparison between ArrayList and HashMap using string as the key and integer as the value. Another graph is the comparison between ArrayList (single object) and HashMap using integer as the key and string as the value. The final graph is the comparison between ArrayList and HashMap using integer as the key and string as the value.

In the comparison between ArrayList and HashMap using ArrayList I and HashMap II, we notice that (Figure 3) up to 100 thousands records ArrayList and HashMap use almost same amount of memory. But then up to 1.2 millions records ArrayList uses significantly less memory than HashMap. And then when the number of records are exceeded 1.2 millions suddenly HashMap uses notably less memory than ArrayList, which is around 10% less memory than ArrayList.

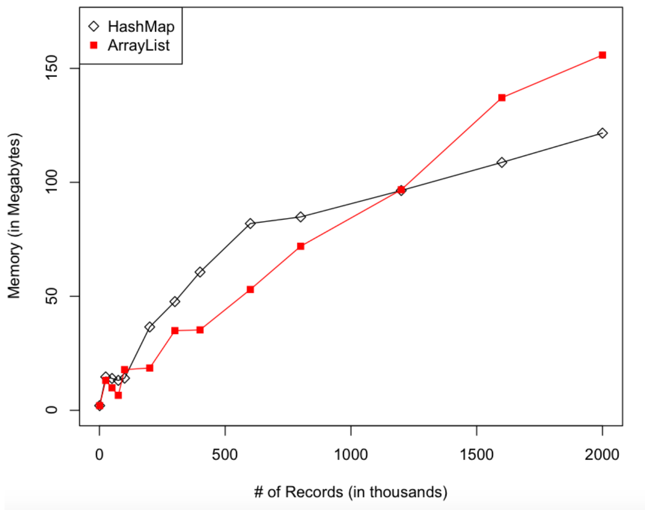


Figure 3. Comparison between ArrayList and HashMap using ArrayList I and HashMap II on memory utilization.

However, In the comparison between ArrayList and HashMap using ArrayList III and HashMap II, we see that (see appendix: Figure 4) for numbers of records up to 100 thousands there is no significant differences between them. But then up to 600 thousands records ArrayList has slightly better memory usage compared to HashMap. And afterward for more than 600 thousands records ArrayList dramatically uses more memory in

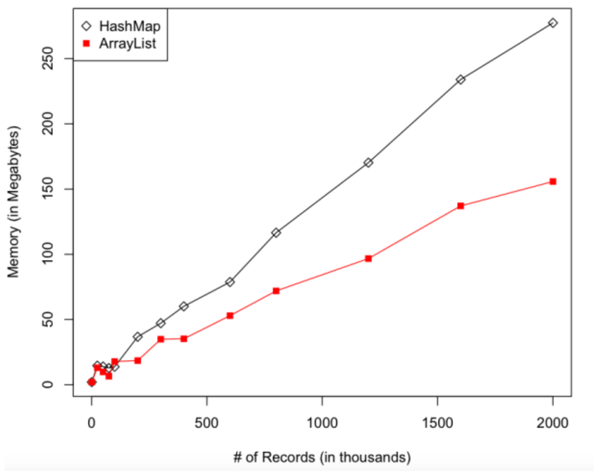


Figure 5. Comparison between ArrayList and HashMap using ArrayList I and HashMap I on memory utilization.

contrast to HashMap, which is around 30% more memory than HashMap. For 2.0 millions of records the difference is around 86 MB, which is around 35% more memory than HashMap.

On the other hand, in the comparison between ArrayList and HashMap using ArrayList I and HashMap I, we figured out that (Figure 5) for the number of records up to 100 thousands there is a negligible difference between them with ArrayList having a slight better memory usages. But then later HashMap uses more memory compared to ArrayList, which is around 40% more than ArrayList. For 2.0 millions of records the difference is around 122 MB, which is around 45% more than ArrayList.

In contrast, in the comparison between ArrayList and HashMap using ArrayList II and HashMap I, we found that (see appendix: Figure 6) there is a negligible differences between ArrayList and HashMap up to 600 thousands records. But, then all of a sudden ArrayList uses considerably less memory than HashMap, which is about 15-20% less memory than HashMap.

From the above discussion, we come to the conclusion that for small databases there are no big differences in memory usage between ArrayList and HashMap. But as databases grow larger, they have some significant differences between them depending on the data types that are used as the key and value.

3.2.2. Overhead Time

When we found that there are some noticeable time differences between ArrayList and HashMap while they store the data, we chose to experiment this time differences, which we considered as the overhead time. The result we got from the experiment are shown below:

Table 2: Overhead Time (in seconds) for various numbers of records (in thousands).

Records	ArrayList		HashMap	
	I	II	I	II
1	0.03790	0.04267	0.04211	0.04906
25	0.18588	0.18737	0.19511	0.20573
50	0.22962	0.25954	0.25740	0.22466
75	0.28051	0.32901	0.32398	0.33557
100	0.32054	0.34742	0.36719	0.39925
200	0.42499	0.45980	0.47691	0.54104
300	0.55336	0.58526	0.61718	0.70898
400	0.65451	0.73265	0.75144	0.94873
600	0.92720	0.99715	1.06129	1.33826
800	1.19559	1.29575	1.32954	1.61566
1200	1.84657	1.85718	1.91518	2.19286
1600	2.34398	2.40009	2.52489	2.81731
2000	2.61896	2.91823	3.30606	3.44197

Note that on the above table, column, ArrayList I, is the measurements of single object. Column, ArrayList II, is measured using key-value pair. And finally columns, HashMap I and HashMap II, are measured using key-value pair where keys are integer and values are string in HashMap I and keys are string and values are integer in HashMap II.

Using the above data another four comparison graphs were plotted: One graph represents the comparison between ArrayList (single object) and HashMap using keys as the string and values as the integer. The following graph represents the comparison between ArrayList and HashMap where keys are string and values are integer. The next graph represents the comparison between ArrayList (single object) and HashMap using key as the integer and value as the string. And the Last graph represents the comparison between ArrayList and HashMap where keys are integer and values are string.

In the comparison between ArrayList and HashMap using ArrayList I and HashMap II, we realize that (Figure 7) for numbers of records up to 100 thousands there is no significant differences between them with ArrayList having negligibly better performance than HashMap. But, then suddenly ArrayList seems to have a better overhead time over HashMap. For 2.0 millions records the difference is approximately 0.8s.

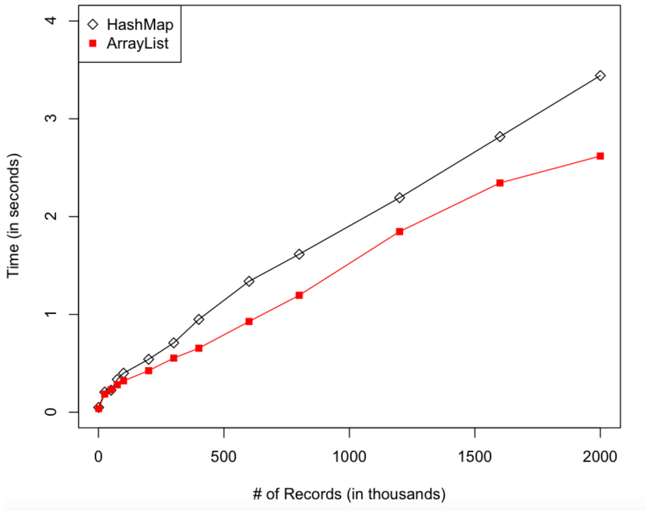


Figure 7. Comparison between ArrayList and HashMap using ArrayList I and HashMap II on overhead time.

Meanwhile, in the comparison between ArrayList and HashMap using ArrayList II and HashMap II (see appendix: Figure 8), the scenario is quite similar as Figure 7. For number of records up to 100 thousands, the difference is trivial. But then later ArrayList has steadily less overhead time than HashMap.

However, in the comparison between ArrayList and HashMap using ArrayList I and HashMap I, we see that (Figure 9) for number of records up to 1.6 millions there is minor differences between them with ArrayList having trivially better overhead time than HashMap. However, after 1.6 million records unexpectedly HashMap consumes significantly more time than ArrayList.

In contrary, in the comparison between ArrayList and HashMap using ArrayList II and HashMap I (see appendix: Figure 10), for number of records up to 1.6 millions both seem to have same overhead time. But, then suddenly after 1.6 millions records ArrayList has steadily less overhead time than HashMap.

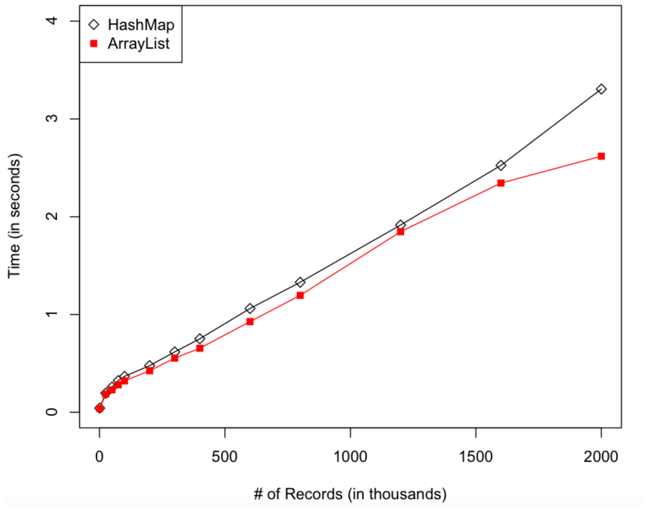


Figure 9. Comparison between ArrayList and HashMap using ArrayList I and HashMap I on overhead time.

From the above discussion, we understand that for smaller datasets there is no immense variance in overhead time between ArrayList and HashMap. But as datasets grow bigger, ArrayList takes moderately less overhead time than HashMap.

3.2.3. Access Time

In the Access time experiment, we noted drastically different behaviors. We noticed that when the numbers of records are larger than 0.1 millions, there is a noticeable difference between ArrayList and HashMap. The result we got from the experiment are shown below:

Table 3: Access Time (in seconds) for various numbers of records (in thousands).

Records	ArrayList		HashMap	
	I	II	I	II
1	0.0002218	0.039325	0.000015	0.042853
25	0.0320334	0.267937	0.000011	0.302271
50	0.0088436	0.560700	0.000010	0.466567
75	0.0089284	0.785223	0.000010	0.683926
100	0.0099176	1.055507	0.000010	1.001105
200	0.0083014	2.285915	0.000009	2.182070
300	0.0082658	3.430187	0.000009	3.516207
400	0.0086292	4.571302	0.000009	5.013680
600	0.0102064	7.157854	0.000013	6.907536
800	0.0125412	10.12845	0.000009	7.031599
1200	0.0134796	14.06706	0.000009	7.189533
1600	0.0152462	18.29040	0.000011	7.261722
2000	0.0151832	22.79678	0.000009	7.039231

Note that on the above table columns, ArrayList I and HashMap I, represent the time they took to access the last index/key. And columns, ArrayList II and HashMap II, represent the time they took to randomly access the data.

Lastly two more comparison graphs were plotted based on the above data. One graph represents the comparison between ArrayList and HashMap for the last index/key access time. Another graph represents the comparison between ArrayList and HashMap for the random access time.

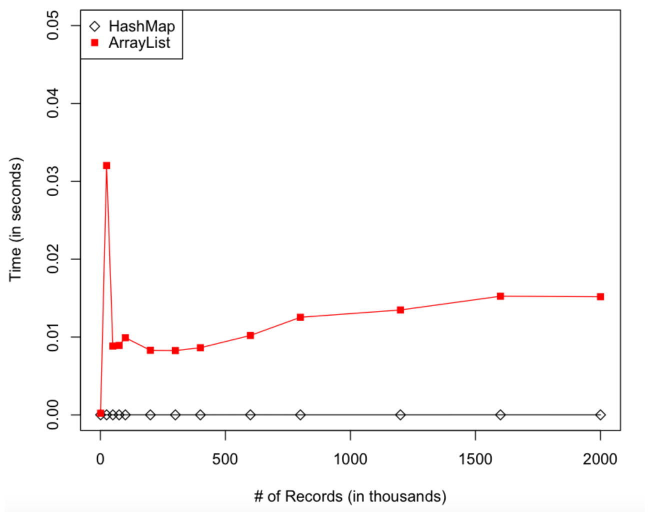


Figure 11. Comparison between ArrayList and HashMap using ArrayList I and HashMap I on access time.

In the comparison between ArrayList and HashMap on Access Time, we found that (Figure 11) for the number of records up to 25 thousands there is a trivial difference between them. But, then speedily up to 50 thousands records ArrayList consumes extremely more times compared to HashMap, which is near 3000 times more than HashMap. And then for more than 50 thousands records the access time of ArrayList significantly dropped down. But still it provides significantly higher access time than HashMap, which is approximately 1500 times higher than HashMap.

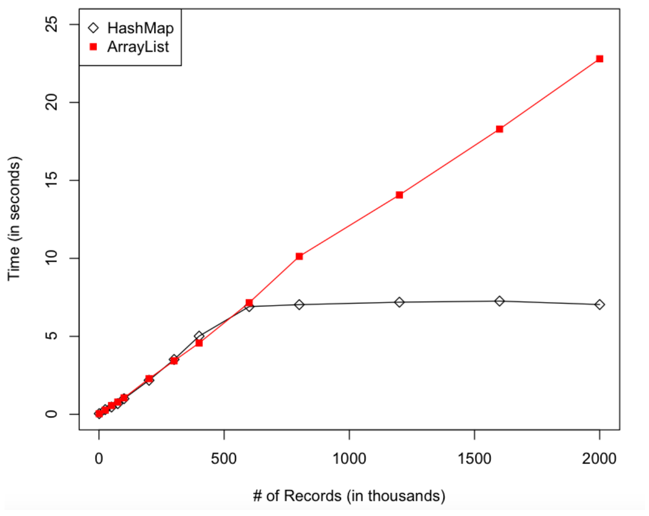


Figure 12. Comparison between ArrayList and HashMap using ArrayList II and HashMap II on access time.

On the contrary, in the Comparison between ArrayList and HashMap using ArrayList II and HashMap II, the picture (Figure 12) is quite different. For number of records up to 600 thousands there are no huge inequalities between them. Both seem to have same access time. But then after 600 thousands HashMap provides drastically better performance than ArrayList, which is about 3 times better access time than ArrayList.

From the above discussion, we finally realize that HashMap provides radically better performance than ArrayList for finding a key from the map. And for the random access time, up to 600 thousands of records there are no massive dissimilarities in access time between ArrayList and HashMap. But as the datasets increase and exceed to 600 thousands records, HashMap provides significantly better performance than ArrayList, which is about 3 times better access time than ArrayList.

4. CONCLUSION

In this paper, we have analyzed and compared Memory Usage, Overhead Time and Access Time on ArrayList and HashMap, two popular java collection classes. After evaluating the experiments we have come to the following conclusions: (1) In general, for number of records up to 0.1 million there are no huge disparities between ArrayList and HashMap except the access time for finding the last index/key, where HashMap provides expressively better performance than ArrayList. (2) As the datasets increase and number of records exceed to nearly 0.1 million records ArrayList uses approximately 20-40% less memory than HashMap if keys are integer and values are string. (3) As the datasets increase and number of records exceed to nearly 1.0 million records HashMap consumes roughly 10% less memory than ArrayList if keys are string and values are integer. (4) Up to 0.6 million records there are no big differences between ArrayList and HashMap for the randomly access to the data. But, later when the number of records exceed to 0.6 million HashMap provides significantly better performance than ArrayList, which is approximately 3 times better than ArrayList.

5. ACKNOWLEDGMENTS

I would like to thank to Dr. Mingrui Zhang, professor of computer science dept. at Winona State University for guiding me to complete this experiment.

6. REFERENCES

[1] Atkinson, Kevin. "SCOWL (And Friends)." SCOWL (And Friends). SCOWL (And Friends), n.d. Web. 16 Mar. 2016. <http://wordlist.aspell.net/>.

[2] Bailey, Chris. "From Java Code to Java Heap." *From Java Code to Java Heap*. IBM, 29 Feb. 2012. Web. 12 Feb. 2016.

[3] Cormen, Thomas H. Introduction to Algorithms. Cambridge, MA: MIT, 2001. Print.

[4] Gray, Simon (Simon James McLean). *Data Structures in Java: From Abstract Data Types to the Java Collections Framework*. Boston: Pearson Addison-Wesley, 2007

[5] Hunt, John. A Beginner's Guide to Scala, Object Orientation and Functional Programming. Cham, Switzerland: Springer, 2014.

[6] McAllister, William. *Data Structures and Algorithms Using Java*. Sudbury, MA: Jones and Bartlett, 2009. Print.

[7] Nick Mitchell and Gary Sevitsky. 2007. The causes of bloat, the limits of health. In Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications (OOPSLA '07). ACM, New York, NY, USA, 245-260. DOI=http://dx.doi.org.wsuproxy.mnpals.net/10.1145/1297027.1297046

[8] Reitbauer, Alois, Klaus Enzenhofer, Andreas Grabner, and Michael Kopp. "Java Memory Management." *How Garbage Collection Works*. Dynatrace, n.d. Web. 23 Feb. 2016.

APPENDIX

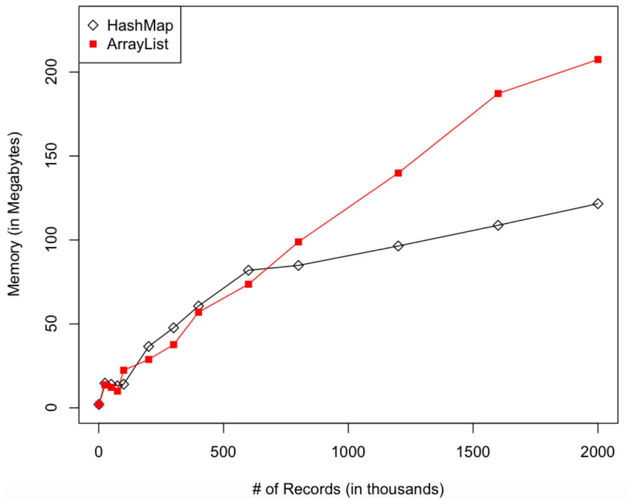


Figure 4. Comparison between ArrayList and HashMap using ArrayList III and HashMap II on memory utilization.

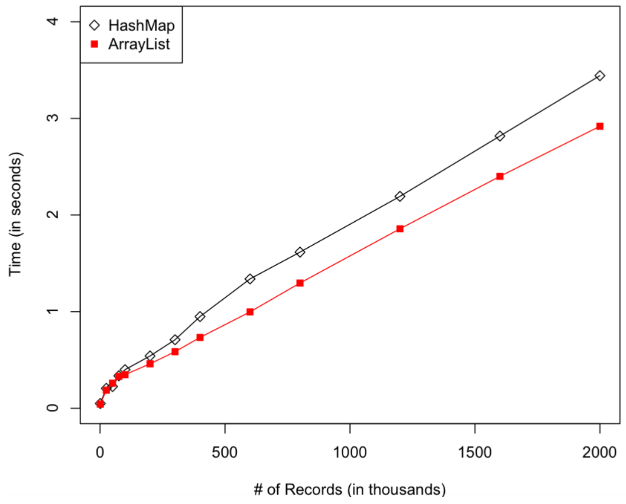


Figure 8. Comparison between ArrayList and HashMap using ArrayList II and HashMap II on overhead time.

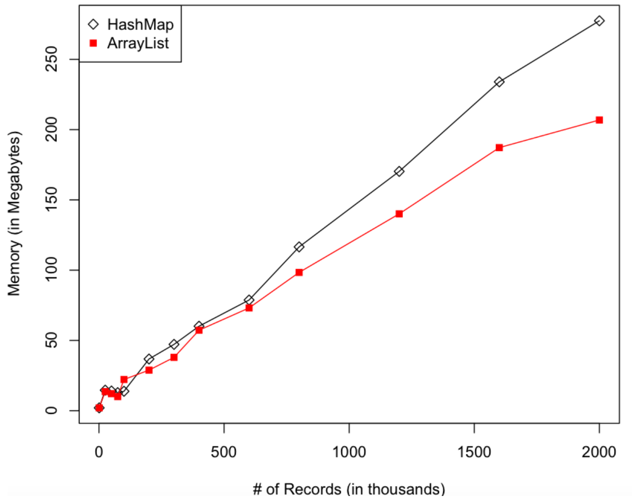


Figure 6. Comparison between ArrayList and HashMap using ArrayList II and HashMap I on memory utilization.

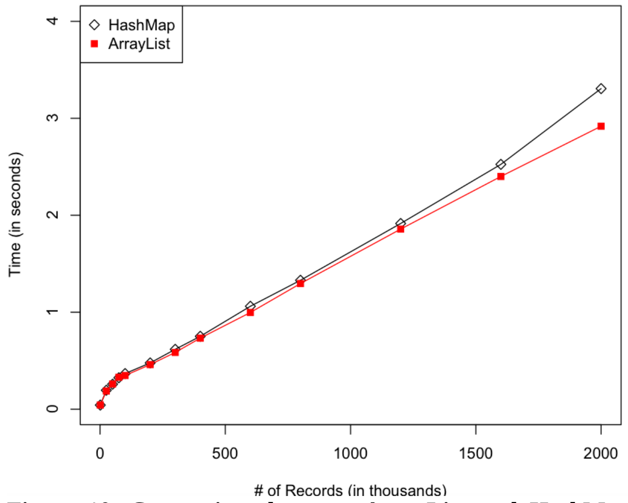


Figure 10. Comparison between ArrayList and HashMap using ArrayList II and HashMap I on overhead time.

Speed index and critical path rendering performance for isomorphic single page applications

Malek Hakim
Winona State University, Computer Science Dept.
175 W Mark St.
55987, Winona, MN, USA
+1(507) 457-5000

hakimelek@gmail.com

ABSTRACT

A current debate in web technology is about whether web apps should be rendered on the client or on the server. Recently, a new approach of using both techniques combined has been developed by rendering the first page load on the server, then rendering the rest of the content on the client to provide an interactive experience for users of a single page app. This approach, called isomorphic Javascript web development, involves running the same Javascript code on the client and the server. In this paper, we measure the speed index and critical path rendering performance of this new technique compared to that of traditional single web apps. We test two versions of a React app delivering a view of a table filled with simulated data. The first version supports server-side rendering while the second does not. The test runs on "WebPagetest," a tool to measure and analyze the performance of web pages. The results show that isomorphic rendering performs better than the traditional approach because it saves the browser the time it takes to render the first views of the website.

General Terms

Human Factors, Web Performance.

Keywords

Javascript, Isomorphism, critical rendering path, code reuse, page load performance, server-side rendering, single page apps.

1. INTRODUCTION

It is important to focus on how web applications behave in terms of loading time in order to provide a greater user experience. A Bing study found that a 10ms increase in page load time costs the site $250K in revenue annually because the improvement in user experience [8]. Optimizing the critical rendering path refers to prioritizing the display of content that relates to the current user action [3]. A study by Amazon showed that an increase of 1.0s in their website page load had an impact of increasing their revenue of $1.6 billion per year [6]. Speed is obviously becoming a critical feature to consider when building web apps.

There are many ways to consider speed in page loading. Testing the true user experience in rendering pages and showing the content that fulfills users goal can be sometimes tricky. Users can wait for a page to fully load until they will be able to see the relevant content, or start interacting with the page once the content they were waiting for gets displayed on the screen. Previous research showed that in order to keep a tasked focused user, the relevant information needs to be displayed on the screen within 1000ms [9] and that is a challenging problem for web developers. The time it takes for a page to be completely loaded generally exceed the time period mentioned earlier. In fact, in average the time it takes for any single page request, without considering caching, going from the DNS lookup, TCP connection with the three-way handshake, the HTTP request generally exceeds the 500ms for a 3G/4G connection [5]. Optimizing the server processing time, downloading time and rendering time either on the client or the server is critical.

In this paper, we focus on the rendering part. Nowadays, there are different approaches to building web application and rendering techniques differ from a full stack technology to another.

1.1 Server-side Rendering

The most popular technique for serving web pages is server-side rendering. Frameworks like ASP.NET, Ruby on Rails and Django endorse server-side rendering and are currently the most popular tools being used on the web [6]. This technique allows pages to be pre-rendered on the server. Users do not need to wait for the Javascript interpreter, as introduced by client-side rendering, to build the application on the browser and bind the data to the HTML in order to see the content on the page. It generally also has the advantage to provide a better Search Engine Optimization (SEO) since pages can easily be crawled. Since client-side rendering requires the page to be rendered on the browser, crawlers index the pre-rendered page, which turns out to be blank since the Javascript does not get executed. Making requests to the server without having to reload the page is limited through AJAX requests. It is a way for the client to communicate with the server through sending and receiving information in a variety of formats thanks to its "asynchronous" nature. Moving from one page to another usually introduces page reload. Figure 1 (at the end of the paper) shows the

Proceedings of the 16th Winona Computer Science Undergraduate Research Seminar, April 27, 2016, Winona, MN, US.

modules that the client and the server are handling. Routing and view rendering occur exclusively on the server. When a user first requests a web page, the server handles the request and generates a rendered HTML file to be sent as a response in addition to the basic CSS, the Javascript code, and the images. The browser will then handle the response and paint the page on the screen; no additional networking is needed until the next user request.

1.2 Client-side Rendering and Single Page Apps

The client-side rendering technique has recently become popular. In this method, the HTML is rendered on the browser using a Javascript payload sent by the server. The Javascript community has developed new frameworks for this method and many web developers have started building Single Page Applications (SPA) in this way. Apps like Gmail and Google Maps are classic examples of a single page app [11]. They let the user interact with the website without having to refresh the page. Today, most of the popular social networks like Facebook, Twitter, GitHub and Flickr are examples of SPAs [2]. As of 2015, 69% of the content on the web is created dynamically on the browser [6]. These apps are more interactive and more pleasant to use. They are also faster because the HTML, CSS, and Javascript code are loaded once throughout the life span of the app, keeping communications with the server to a bare minimum. It is also less heavy on the network bandwidth since communication after the first load with the server is using only data through XML or JSON files.

However, SPAs have the disadvantage of not being very suitable for SEOs and do not have the benefit of allowing pages to be indexed by search engines. Crawlers usually index "blank" pages, for example, in a news article app since the first page is loaded empty waiting for the Javascript to fetch the data from an API server. It also takes more time for content to load before the user interface can be rendered. Another problem that arises with the use of SPA is maintaining two similar business logic programs written in different languages, using any programming language on the server, and Javascript on the client. Figure 2 illustrates this problem. Once the code is loaded on the browser, the page remains blank until the Javascript renders the entire page. Usually functions with the same logic are written twice: once because it is handled by the front-end Javascript and once for validation for the server.

1.3 Supporting Server-side rendering in Isomorphic Single Page Apps

SPAs present an advantage from a user experience standpoint: There is no need for a page refresh, the state of the app is preserved, and the app is more interactive. However, they introduce challenging problems related to SEOs, performance and maintainability. In order to overcome these problems, web developers can take advantage of a recent Javascript runtime built on the Chrome's V8 Javascript engine called NodeJs [10]. It allows developers to use Javascript as programming languages on the server and provide ways to create web servers that perform networking operations and handle file system I/O. It makes it easier to create more maintainable code that can be run both on the client and the server. This approach also facilitates writing code once. The same codebase for view rendering for example is executed during the initial page load on the server then executed on the browser to create a consistent single page experience.

Most of the new implementations of isomorphic web apps use the first approach of server-side rendering, in order to render the first page bound with data, plus a Javascript bundle that will be used for the client to keep performing the business logic and fetching data through user interaction without having to reload the page. Libraries like React and Rendr are pioneers in implementing this technique. Web crawlers able to index pages easily even with having Javascript disabled in the browser [12]. Thanks to the isomorphic nature of web apps that run on NodeJS, in web development the business logic can now be shared on both the client and the server. The code can be written once and executed as needed on both ends. Figure 3 illustrates the code base of the isomorphic approach where view rendering occurs on the client and on the server. DOM manipulation, animation and form validation are handled on the browser, while persistence is handled on the server. The client and the server depending on the program flow share the rest of the business logic that covers all the app functionality.

The sequence diagram in Figure 3 shows the sequence of events when a user requests a new page. The browser receives an initial render HTML with the CSS and a Javascript bundle that will build the client-side app for further navigation and user interaction. However, compared to the previous model, the user will receive a UI ready app without having to wait for the Javascript to re-render the DOM tree and display the content on the page. After the app is done building on the client, it makes requests for the resources the user is asking for. This requires an additional round-trip with the server or the API server.

Isomorphic Javascript apps seem to mix the two rendering techniques to optimize the user experience for displaying the relevant content to the user quicker.

2. METHODS

2.1 Metrics

Server-side rendering in SPAs is considered an optimization in the critical rendering path. We examined the following metrics [13] to compare the traditional approach, which is limited to client-side rendering, to the optimized approach using isomorphic Javascript.

2.1.1 Page Weight

Page Weight represents the page size in kilobytes. It combines the size of the total web app file that are received by the browser.

2.1.2 Visually Complete

Visually Complete is the time when the videos frames are 100% complete when video recording is enabled during the page loading.

2.1.3 Start Render

The Start Render time is the first point in time that something was displayed to the screen. Before this point in time the user was staring at a blank page. This does not necessarily mean the user sees the page content. It could just be something as simple as a background color but it is the first indication of something happening for the user.

2.1.4 Speed Index
The Speed Index calculates how "complete" the page is at various points in time during the page load. The lower the speed index is, the better. It is expressed by the following equation [14]:

$$\text{Speed Index} = \int_0^{end} 1 - \frac{VC}{100}$$

end = end time in milliseconds
VC = % visually complete

Speed Index uses video recording during the page load and captures screenshot through the time (10 frames each second in the WebPagetest tool implementation used). A histogram of colors of each of the frames is then generated and takes the overall distribution of colors on the page. The difference of the histogram for each frame in the video versus the first histogram is compared to the baseline to determine how "complete" that video frame is.

2.1.5 Dominteractive
Dominteractive is the amount of time spent between first known startup of the application platform and when the UI is interactive regardless of view. Note that this does not require that the UI is done loading, but is the first point at which the customer can interact with the UI using an input device.

2.1.6 First Byte
The First Byte time is the time from when the user starts navigating to the page until the first bit of the server response arrives. The bulk of this time is usually referred to the "back-end time" and is the amount of time the server spends building the page for the user.

2.2 Environment Setup and Test Case
For the purpose of this experiment, a web app with two variances was served from the same server on different ports. It is an open source Web application using NodeJS, Express and ReactJS [3]. It is a basic example of isomorphic Javascript. It uses a Griddle React component in order to display a basic table filled with simulated data. The table contains 200 rows and 7 columns with the ability to sort through the columns. The Javascript running on the browser handles the sorting of the columns. Once the page is fully loaded and the DOM becomes interactive, the table can be sorted. The first version was the original one using ReactJS rendering on the server the first page load. The second version is similar except that the server rendering code was deleted and the app served only the Javascript bundle that render the view on the browser.

The following code shows the HTML code that will be used to render the page:

```
<div id="react-main-mount">
  <%- reactOutput %>
</div>


<script src="/main.js"></script>
```

"reactOutput" is a reference to a React component which serves as a rendering template on the server. Deleting <%- reactOutput %> would result to an exclusive client-side rendering handled by the Javascript bundle "main.js" attached to the HTML file. The Javascript file is also referencing the same React component that "reactOutput" is referencing. This implementation illustrates the isomorphic nature of React.

React is using many tricks to make page load performance faster and probably its most popular feature is the use of a virtual DOM that reflects the actual DOM. React makes changes into the virtual DOM and when it is ready, it batches DOM updates through the use of difference algorithms. For the matter of the measurement of page load performance for isomorphic web apps, we do not focus on that feature and are varying only the rendering mode from the server and the client to concentrate our study on the isomorphic nature. We also do not compare our results to the optimal page load time. We only compare the two approaches tested.

One of the popular tools to test the metrics mentioned earlier is the "WebPagetest" tool. It is a web performance tool that uses real browsers to access web pages and collect timing metrics. We performed the test 10 times and took the median of the results. All the tests were run on a 3G network using Chrome as the default browser.

3. RESULTS
The results for the single page app tested show that the browser takes the same time as the client-side approach to receive the first bytes when using isomorphic rendering. However, the Start Render time is delayed compared to the client-side approach. By looking at the median of the first bytes, the delay experienced by the isomorphic approach is explained by the fact that it takes more time to process the view rendering on the server.

We notice a 0.4 seconds difference in the Start Render time; it is explained by the fact that the isomorphic approach for the HTML is much heavier. However, the isomorphic approach has the advantage of having a faster Visually Complete time and a slower Speed Index. The isomorphic approach takes less than 9.25 seconds to render a visually complete page from the time of Start Render compared to the client-side approach, which takes 13.25 seconds to render the table and display the complete page from the time of Start Render.

Figure 4 illustrates the visual completeness of the two approaches and shows the progress of the rendering on the screen. The Visually Complete metric and the Speed Index are always the same value for the isomorphic approach. It is explained by the fact that the page is not progressively rendered since the complete HTML was served with the data during the initial page load.

Table 1. Results of the tests run using "WebPagetest"

Approach	Page weight	Visually Complete	Speed Index	Dominteractive	First Byte	Start Render
Client-side rendering	2,106 KB	13.25s	5982.5	12.9s	0.839s	3.437s
Isomorphic rendering	2,390 KB	3.9s	3900	14.77s	0.841s	3.888s

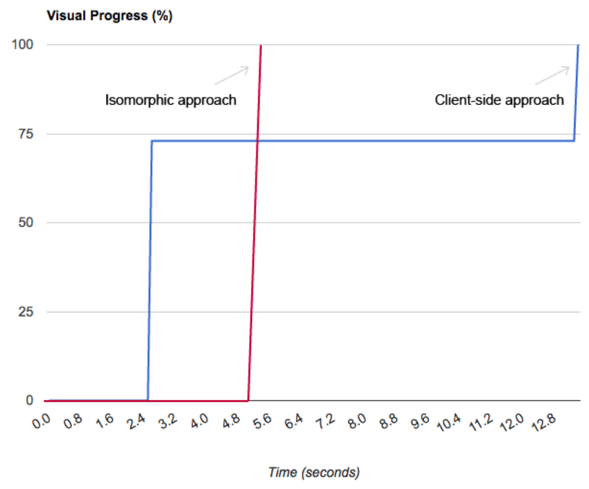


Figure 4. Visual completeness vs. time for client-side rendering and isomorphic rendering

We notice a difference in time in favor of the client-side approach, which has a reduced time for the Dominteractive metric. On the other hand, isomorphic is much better in optimizing the critical rendering path. Content is displayed faster but the page takes longer to be interactive, i.e., not yet the user is able to interact with the displayed page. In our case, filtering the table is an example of user interaction. Also, we notice that the page weight is heavier for the isomorphic approach because it carries a heavier pre-rendered HTML file. In this case, mobile browsers may have problems supporting larger files because of the limitations in CPU usage and memory storage of mobile devices.

Another factor that determines which method is better is the intent of the app. If it is an app for content delivery, like a news web app, showing content very quickly is a necessity. If it is an app where users need to interact heavily with the app through button clicking, drag and drop, etc., then waiting for the app to fully load helps increase the perceived performance. A possibility of adding a progress indicator like a spinner or a loading bar can be an alternative in this case to keep the user task focused

4. CONCLUSION

Isomorphic Javascript is a promising approach, especially with the increasing engagement of the NodeJS and Javascript open source community. In this paper, we have covered the evolution and different approaches of rendering content on the server and on the browser. We claimed that the isomorphic approach would have a faster load time and lower Speed Index and, after performing an experiment on a single page app with two variances to support client-side rendering only and isomorphic rendering, we validated our hypothesis. We also discovered that the faster the page is visually loaded, the more time it takes to become interactive. In addition, the results showed that the isomorphic approach causes Page Weight to be higher than the traditional approach.

5. ACKNOWLEDGMENTS

My thanks to Dr. Joan Francioni and Dr. Mingrui Zhang for their reviews and contributions to this research work.

6. REFERENCES

[1] Built With, "Framework Usage Statistics," [Online]. Available: http://trends.builtwith.com/framework. [Accessed: 4- Apr- 2016].

[2] D. Pupius, (2013, Jan.). "Rise of the SPA,". [Online]. Available: https://medium.com/@dpup/rise-of-the-spa-fb44da86dc1f. [Accessed: 4- Apr- 2016].

[3] D. Walles. "What is Isomorphic/Universal Javascript?". [Online]. Available: https://github.com/DavidWells/isomorphic-react-example. [Accessed: 18- Apr- 2016].

[4] I. Grigorik, 'Google Developers', Critical Rendering Path. [Online]. Available: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/?hl=en. [Accessed: 27- Mar- 2016].

[5] I. Grigorik, "Optimizing the Critical Rendering Path," presented at the O'Reilly Velocity Conference, Santa Clara., CA. 2013. [Online]. Available: https://docs.google.com/presentation/d/1IRHyU7_crIiCjl0Gvue0WY3eY_eYvFQvSfwQouW9368/present?slide=id.p19. [Accessed: 4- Apr- 2016].

[6] K. Eaton, 'Fast Company', How One Second Could Cost Amazon $1.6 Billion In Sales, 2012. [Online]. Available: http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales. [Accessed: 27- Mar- 2016].

[7] K. Probst, B. Johnson. "Making Ajax Crawlable," [Online]. Available:

http://www.scriptol.fr/ajax/SMX_East_AJAX_Proposal.pdf . [Accessed: 4- Apr- 2016].

[8] Microsoft, 'IEBlog', HTTP/2: The Long-Awaited Sequel, 2014. [Online]. Available: https://blogs.msdn.microsoft.com/ie/2014/10/08/http2-the-long-awaited-sequel/. [Accessed: 27- Mar- 2016].

[9] M. Kearney, 'Google Developers', The RAIL Performance Model. [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/profile/evaluate-performance/rail?hl=en. [Accessed: 4- Apr- 2016].

[10] Nodejs Home Page. [Online]. Available: http://nodejs.org. [Accessed: 4- Apr- 2016].

[11] P. Lenssen, (2008, Jun.). "Kevin Fox of Gmail & FriendFeed on User Experience Design,". [Online]. Available: http://blogoscoped.com/archive/2008-06-02-n56.html. [Accessed: 4- Apr- 2016].

[12] S. Brehem, (2013, Jan.). "Isomorphic Javascript: The future of web apps". [Online]. Available: http://nerds.airbnb.com/isomorphic-javascript-future-web-apps/. [Accessed: 4- Apr- 2016].

[13] WebPageTest Documentation. [Online]. Available: https://sites.google.com/a/webpagetest.org/docs/. [Accessed: 4- Apr- 2016].

[14]

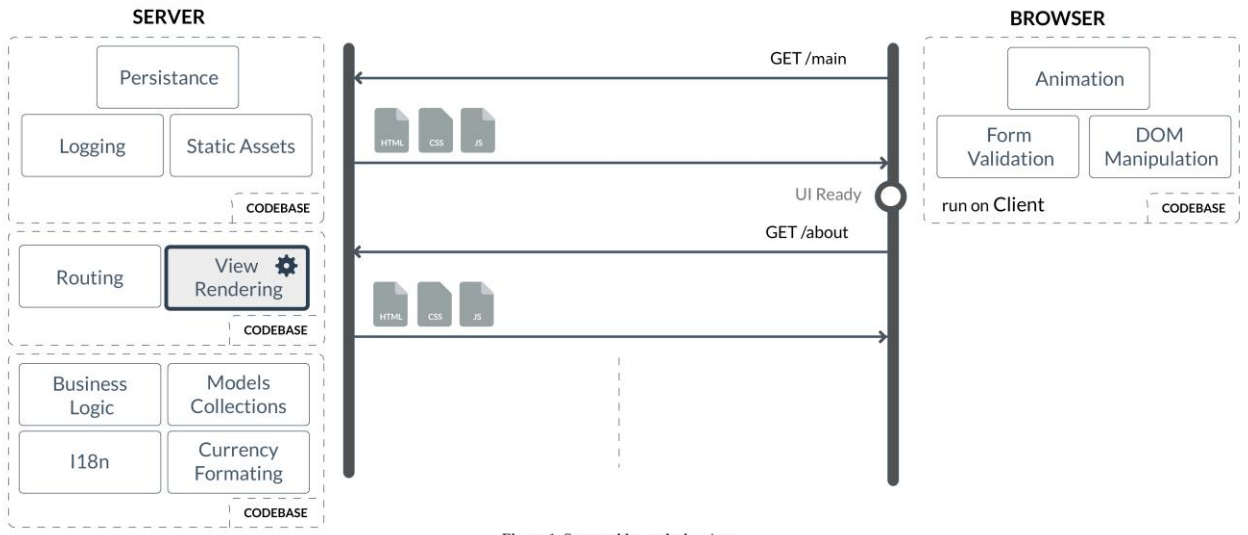


Figure 1. Server-side rendering Apps

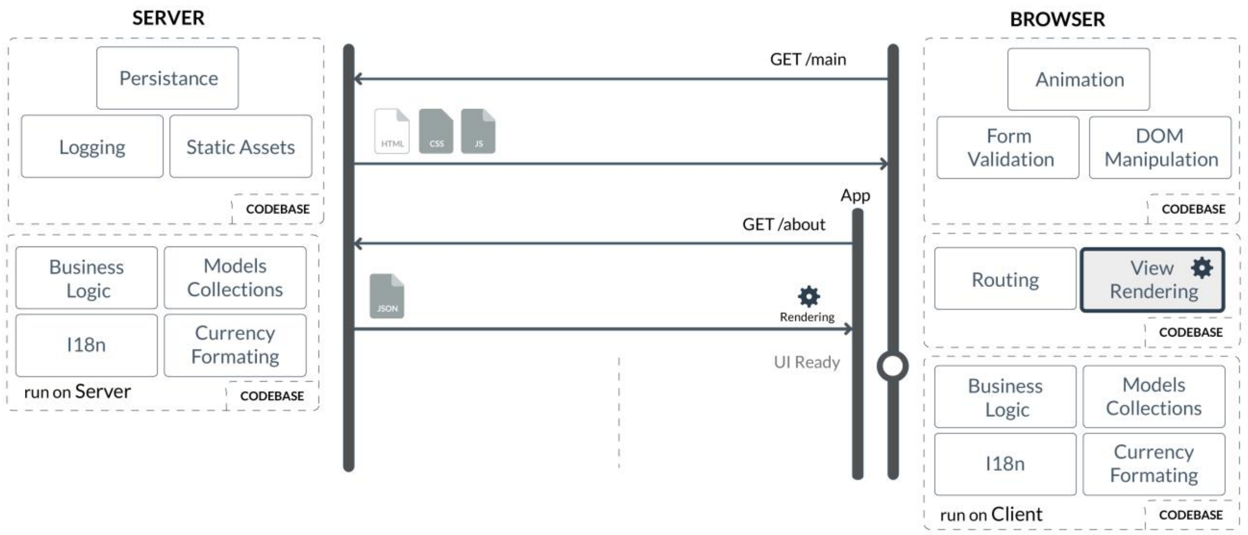


Figure 2. Client-side rendering Apps

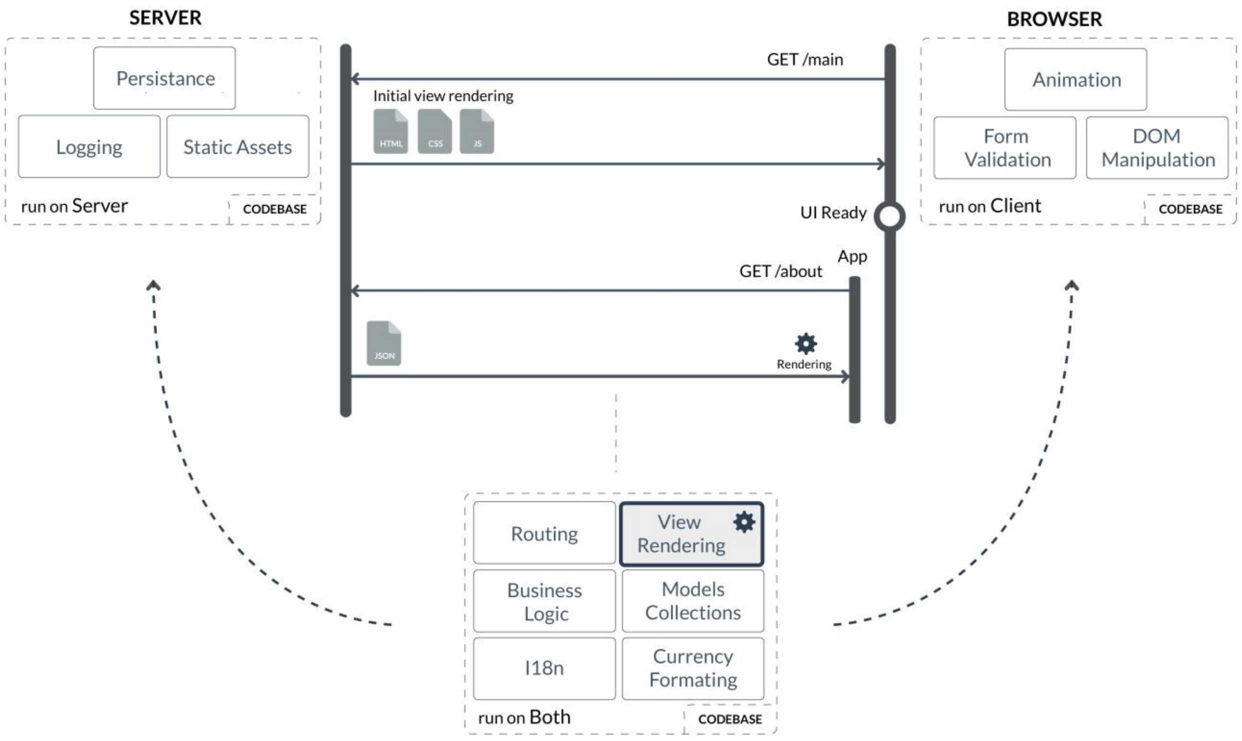


Figure 3. Isomorphic Javascript Apps