

# NETWORK RELIABILITY API FOR FIREFOX OS

## Improving Efficiency Through Analysis of Network Metadata

May 26, 2015

John Zeller  
Pok Yan Tjiam  
Jonathan McNeil

## Contents

# 1 Introduction

This project was requested by Mozilla, specifically the team working on Firefox OS. Our customer during the course of our project was Dietrich Ayala, a Project Manager working on Firefox OS and a member of the Mozilla family since February 2006. Dietrich's role in our project took many forms, and evolved as what we needed did so. Initially, he helped on board us to the Firefox OS platform, providing us with plenty of documentation to wrap our heads around, and once we have devoured this information, he was quick to setup meeting with members from the Firefox OS networking team. Throughout the year Dietrich was always available via IRC, Skype, Email, and Cell, whenever we needed him. And if he could not answer a question for us, he knew who could.

The members of our team were John Zeller, Pok Yan Tjiam, and Jonathan McNeil. John Zeller, having worked with Mozilla for over a year as an intern and contractor, acted as a de facto team lead during the course of the project. Pok Yan Tjiam worked on \_\_\_\_\_. Jonathan McNeil worked on \_\_\_\_\_.

The purpose of this project was to help work towards giving developers a tool that allows them to avoid the typical environment agnostic network request paradigm. As it currently stands, most developers create apps with little to no information about the quality of the network. In their apps, they typically run network requests in a dumb loop, which simply tries continually until the request succeeds. This is wasteful in terms of both battery and data, and overloads an often times already overloaded network.

Firefox OS is not a direct competitor to the well known giants in the mobile OS space, but is aimed at being a good solution for the next 4 billion people coming online; the majority of which are coming online in developing countries where network infrastructure is literally decades behind, having a profoundly negative impact on user experience. In many cases poor network conditions can even render some applications non-functional. Aside from out-dated networks, some areas of these developing countries also have unreliable electrical grids, which can make charging a phone take days for many users living with the most unreliable grids. And this problem is only growing worse, at an ever growing pace. In the next 30 days, India alone is expected to account for 5 million new internet users.

## **2 Requirements Document**

### **2.1 Original**

Here is our original requirements document, featuring the details of our project as we understood them at the time it was written.

# **CS 461 / 462 / 463**

## **Client Requirements Document**

### **Task Scheduler API for Firefox OS**

**Team Name:** FxOSU

**Team Members:**

John Zeller	(zellerjo@onid.oregonstate.edu)
Jonathan McNeil	(mcneilj@onid.oregonstate.edu)
Pok Yan Tjiam	(tjiamp@onid.oregonstate.edu)

**Client/Sponsor/Mentor:**

Dietrich Ayala  
Organization: Mozilla  
Phone: 503-512-9252  
E-mail: dietrich@mozilla.com

**Introduction to the problem:**

Firefox OS is an open source operating system for smartphones developed by Mozilla. It is a web-centric system based on Linux kernel. Firefox OS phones have launched in 15 countries around the world. It targets low cost market in developing countries.

**Project Description:**

Developing countries like India usually have poor quality infrastructures. Unreliable network and unreliable electrical grid is a huge problem that affect user experience. The poor network condition makes some of the applications become non-functional. Charging the battery of the phone can even take days for users living among the most unreliable electrical grids.

A solution to this problem is a job scheduler API which could allow the phone to delay a task until it has a good network connection or has a power cord plugged in. This should preserve the functionality of the phones under different network condition and reduce inefficient use of network access. Since all Firefox products share the same core, this API is expected to be deployable in not only the Firefox OS but also in Firefox for Android and Firefox for Desktop.

**Requirements:**

Req. #	Requirement	Status	Comments
1	Analyze existing efforts (ie ServiceWorkers, RequestSync) to determine if they can be integrated with our system.	Pending	
2	The API should integrate with existing efforts wherever possible.	Pending	
3	The prototype API should be written in JavaScript	Pending	
4	The API should be written in C++	Pending	
5	The API should be callable by JavaScript executing in the DOM in a web context.	Pending	
6	The prototype API should be able to queue tasks	Pending	
7	The prototype API should be able to prioritize tasks in the queue based on network, battery and system data.	Pending	
8	The prototype API should be developer configurable, to provide a force execution of a task after a period of time	Pending	
9	The prototype API should be able to function without error on Firefox OS, Firefox for Android, and Firefox for Desktop	Pending	
10	The API should be able to access data on the quality of the network in order to determine if a task should be executed.	Pending	
11	The API should be able to access data on the charging state of the device in order to determine if a task should be executed.	Pending	
12	The API should be able to access data on the battery level of the device in order to determine if a task should be executed.	Pending	

13	The API should be able to access data about system load on the device in order to determine if the device can handle another task.	Pending	
14	The API should passively collect network status information.	Pending	
15	The API should analyze network data to determine patterns in network reliability	Pending	
16	The API should be able to queue and postpone tasks from being dispatched	Pending	
17	The API should be able to prioritize queued tasks based on accessible data about the device and the environment	Pending	
18	The API should have a mechanism for dispatching queued tasks	Pending	
19	The API should be developer configurable, to prioritize application specific tasks	Pending	
20	The API should be developer configurable, to provide a force execution of a task after a period of time	Pending	
21	The API should be able to function without error on Firefox OS, Firefox for Android, and Firefox for Desktop	Pending	
22	The test app for the API should be written to function on Firefox OS, Firefox for Android, and Firefox for Desktop	Pending	
23	The test app for the API should benchmark device resource usage, contrasting use with the API and use without the API	Pending	
24	Write a Web IDL specification for the API implementation	Pending	

## **Versions:**

- Version 1: Create a prototype in Javascript. It does not need to be fully functional.
  - 1.0 Network monitoring functionality
  - 1.1 Process and Power monitoring functionality
  - 1.2 Task Queuing functionality
  - 1.3 Task Prioritization functionality
- Version 2: Implement the components of job scheduler API in C++.
  - 2.0: Network, Power and Process monitoring functionality
  - 2.1: Fully functional job scheduler with task queueing and dispatch functionality
- Version 3: Combine all functionalities into the final implementation of the job scheduler with testing application included.

## **Design:**

This job scheduler API will be put into the network stack of Firefox OS which is called Necko. Necko will be where we obtain all of the necessary data to determine whether a task should be performed. We will then “schedule” tasks based on this information in a prioritized queue. These tasks will then be dispatched when the conditions are appropriate or a predetermined amount of time has passed (we don’t want starvation).

The major components of this API will be systems to monitor network quality and traffic, charging state and battery life, and current system load. There will also be a scheduling system which will place tasks in the queue to be dispatched when the conditions are correct. There will also be a system which prioritizes delayed to prevent overload when good network conditions or the charging state of the device changes.

## **Specific tasks to be undertaken:**

- Network monitoring function
- Power monitoring function
- Process monitoring function
- Developer configuration options
- Task queueing function
- Task dispatch function
- Testing application
- Draft a Web IDL specification document

### **Risk Assessment:**

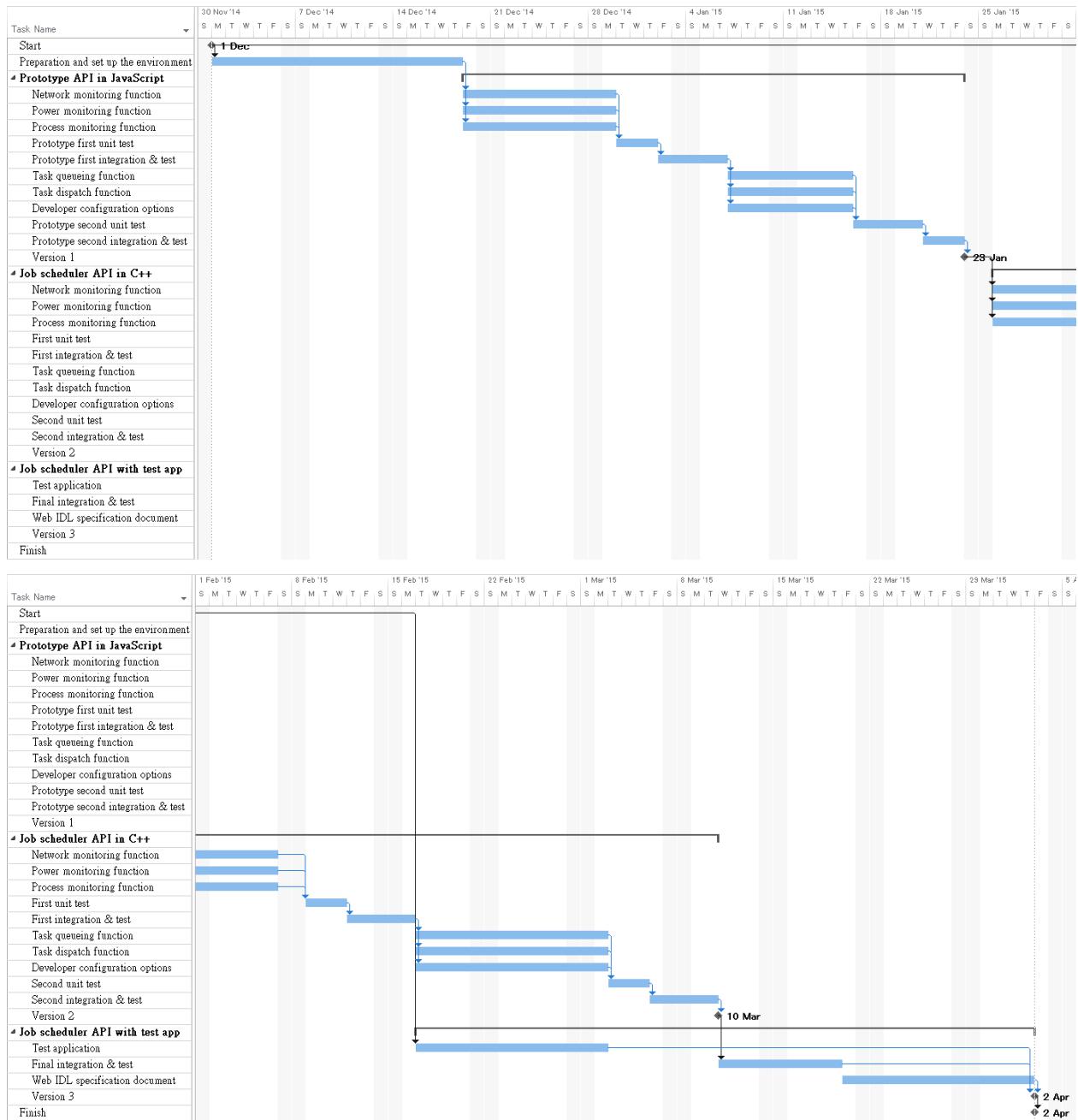
One of the challenges associated with creating an Programming Interface that is designed to minimize power usage is creating the interface in a way that doesn't hurt more than it helps. It could be useful if we could create a functionality that can schedule tasks, but if our method of doing so costs more battery usage then it saves we really haven't solved the problem. This applies to all of the monitoring functionalities which we intend to develop. If the constant monitoring of the devices resources is too costly I fear we may not actually solve the problem, but only provide a useful functionality for developers to use as part of the solution to the problem. This gets even more difficult as we might not even see it happening. We won't be able to tell if our API has real world results until it has been tested.

For any of the challenges associated with this project, both foreseen and unforeseen, there are a couple of options for what to do. Development in the mozilla core is not something that is being done alone. There is a thriving IRC network with developers with far more experience than any of us. The first thing I would do is start a discussion here to see if there are any methods of resolving the issue. Even if a direct solution doesn't present itself, we may be put into contact with someone who can give us more information in the area where we are experiencing the challenge. If a challenge becomes a disaster, the final option would be to sit down with our client and talk about work-arounds or requirement changes. This would be a last resort scenario.

### **Testing:**

Unit tests will be created to test the functionality of the network, power and process monitoring functions and task queueing and dispatch functions separately. After all tests passed and all components integrated together, an integration test will be performed by building an example Firefox OS application to monitor the network and power usage of a Firefox OS device. The application will be used to test the performance of the Firefox OS device under different network conditions and battery charging states, and visualize the effect of the job scheduler in term of efficiency. We will perform the tests before each version released.

## Preliminary Timetable:



**Roles of the different team members:**

The entire team will work together on the test application and Web IDL specification document. For the JavaScript prototype (version 1) and the scheduler API in C++ (version 2), we will divide the API into 6 functionalities in both versions and each of the member is in charge of 2 functionalities. Each member will have equal workload.

**Integration Plan:**

After finished the unit testing on each components and make sure they are all functional, we will integrate them into a single job scheduler API. We will upload our files to Github so that all the member can work together online. After the integration of all parts, we will do an integration test to see if our API works correctly or not.

**References:**

- [https://developer.mozilla.org/en-US/Firefox\\_OS](https://developer.mozilla.org/en-US/Firefox_OS)

**Glossary:**

- Gecko: a layout engine designed to support open internet standards like HTML
- Necko: the network library that provides an extensible API for several layers of networking

## **Signatures**

### **Team members:**

---

John Zeller

---

Date

---

Jonathan McNeil

---

Date

---

Pok Yan Tjam

---

Date

### **Client/Sponsor/Mentor:**

---

Dietrich Ayala

---

Date

## 2.2 Revisions

### 2.2.1 Requirements

As happens in most projects, requirements change, and ours was no different. Here are the changes that were made to our requirements and why.

#	Requirement	What Happened To It	Comments
4	The API should be written in C++.	Modified Changed to "The API should be written in C++ and/or JavaScript"	The reason for this...
5	The API should be callable by JavaScript executing in the DOM in a web context.	Modified First became "The API should be callable by JavaScript executing in the window.", Then modified again to become "The prototype API should be callable by JavaScript executing in a web sandbox."	The reason for this...
6	The prototype API should be able to queue tasks.	Removed New requirement was "The prototype API should be able to access data on the battery level of the device in order to determine if a task should be executed."	The reason for this...
7	The prototype API should be able to prioritize tasks in the queue based on network, battery and system data.	Removed New requirement was "The prototype API should be able to access data on the charging state of the device in order to determine if a task should be executed."	The reason for this...

#	Requirement	What Happened To It	Comments
8	The prototype API should be developer configurable, to provide a forced execution of a task after a period of time.	Modified Changed to "The prototype API should be developer configurable, to provide a level of certainty about network quality."	The reason for this...
9	The prototype API should be able to function without error on Firefox OS, Firefox for Android, and Firefox for Desktop.	Modified Changed to "The prototype API should be able to function without error on Firefox for Desktop."	The reason for this...
10	The API should be able to access data on the quality of the network in order to determine if a task should be executed.	Modified Changed to "The prototype API should be able to access latency-related network information to determine if a task should be executed."	The reason for this...
16	The API should be able to queue and postpone tasks from being dispatched.	Removed New requirement was "The API should be able to access latency-related network information to determine if a task should be executed."	The reason for this...
17	The API should be able to prioritize queued tasks based on accessible data about the device and the environment.	Removed New requirement was "The API should take into account the type of network connection, whether it be wifi, cellular data, etc."	The reason for this...
18	The API should have a mechanism for dispatching queued tasks.	Removed New requirement was "The API should be able to access data about recent tx/rx data."	The reason for this...

#	Requirement	What Happened To It	Comments
19	The API should be developer configurable, to prioritize application specific tasks.	Modified Changed to "The API should be developer configurable, to provide a level of certainty about network quality."	The reason for this...
20	The API should be developer configurable, to provide a force execution of a task after a period of time.	Removed New requirement was "The prototype API should be able to access data about recent tx/rx data to determine if a task should be executed." Then, the requirement was modified to become "The prototype API should be able to see if the device has an internet connection to determine if a task should be executed."	The reason for this...

## 2.2.2 Gaant Chart

Here is the final Gaant chart for how our project timeline actually turned out.

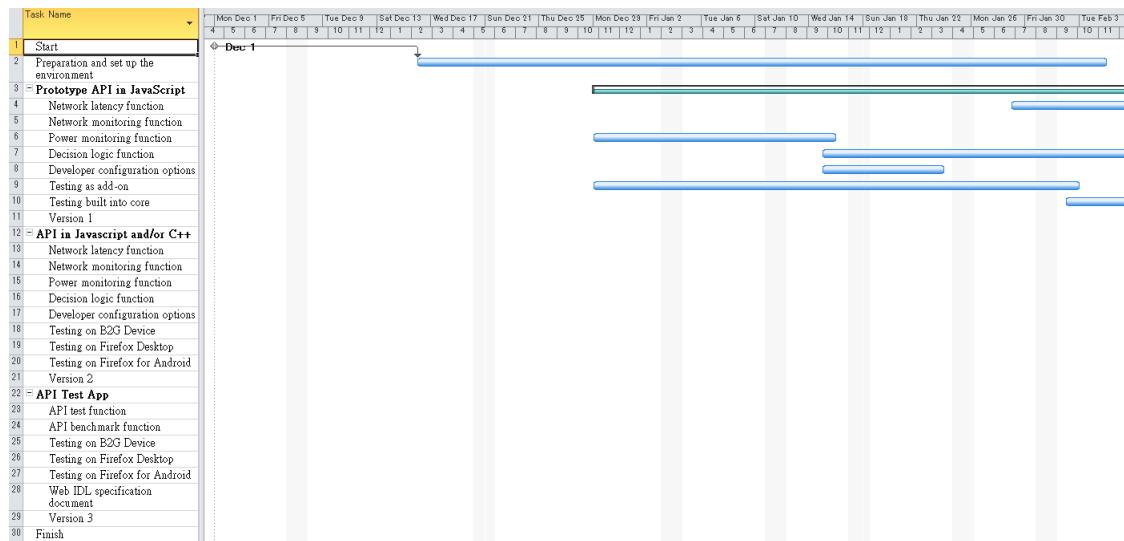


Figure 1: 1 of 3: The final Gaant chart for how our project timeline actually turned out.

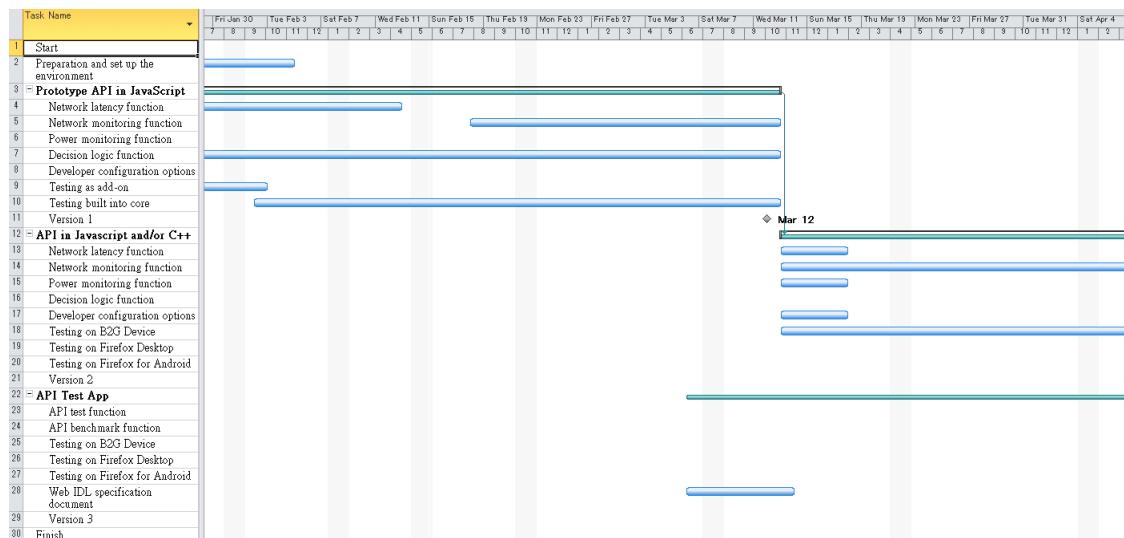


Figure 2: 2 of 3: The final Gaant chart for how our project timeline actually turned out.

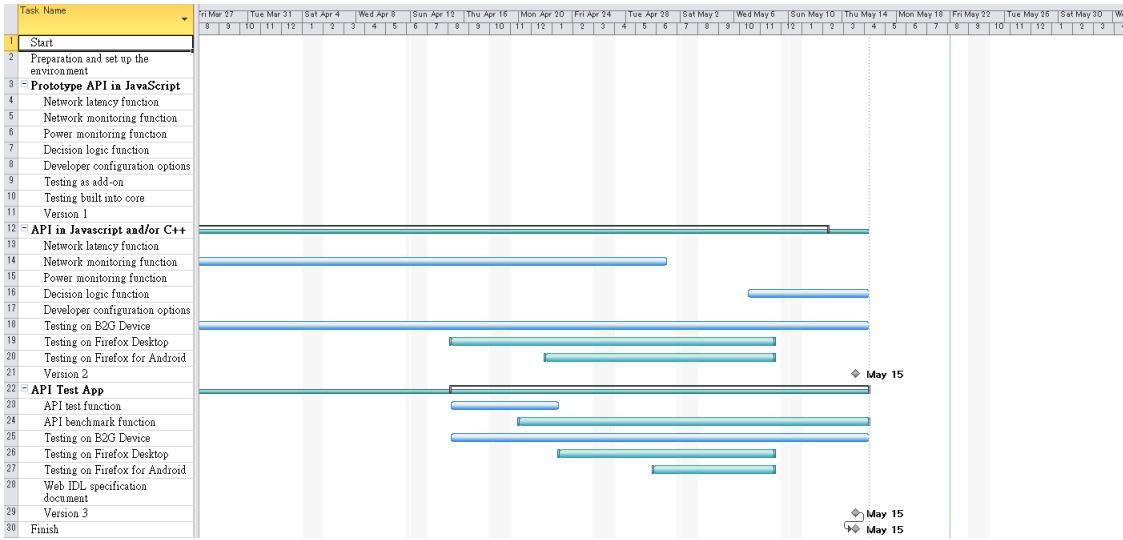


Figure 3: 3 of 3: The final Gaant chart for how our project timeline actually turned out.

### 3 Weekly Team Blog Posts

Friday, October 31, 2014

Fall 2014, Week 5:

Plans for coming week:

- Start working on the requirements documentâ„ў
- Continues to research on Firefox OS

Progress since last week:

- Problem statement finished
- Technology reviewâ„ў finished
- Project blog and project site set â„ўupâ„ў
- Problem statement and technology reviewâ„ўâ„ў uploaded to the project site

This week we set up our project blog and project site using Sharepoint space. The project site is used to store and share documents we made such as the problem statement

and technology reviewâ„“ to the whole team. The project blog is used to keep track of our progress on this project. We posted our progress updates on the project blog every week so that our client can know what we are currently working on.â„“

Friday, November 7, 2014  
â„“Fall 2014, Week 6:

Plans for coming week:

- Continues to work on the requirements document
- Continues research (necko), get familiar with Web IDL specifications

Progress since last week:â„“

- Finished the basic structure of the requirements documentâ„“

We expect to finish the introduction and project description parts of the requirements document, and think of some requirements for this project in the next week. We create our first draft of the requirement document in Word format instead of LaTeX because it is easier for us to edit it online. We stored our requirement document on Google Drive and shared it to the whole team as well as our client so that every one in our team can work on the document at the same time. Also, our client can give us immediate feedback without sending email but simply add a line somewhere on the document. We followed the format provided in the assignment page and finished the basic structure of the requirements document at the end of this week.

Friday, November 14, 2014  
â„“Fall 2014, Week 7:

Plans for coming week:

- Continues to work on the requirements document
- Finish the requirement parts of the requirements document

â„“Progress since last week:â„“

- Finished the introduction part of the requirements documentâ„“

We continue to work on the requirement document in this week. We combined the words FxOS and OSU to FxOSU and decided to use it as our team name. Our progress on the requirement document was pretty slow because we have to do more research on Firefox OS at the same time. At the end of this week we finished the introduction part of the requirements document and filled in some requirements of the project.

Friday, November 21, 2014

Fall 2014, Week 8:

Plans for coming week:

- Finish the whole requirements document
- Get the requirements document signed
- start working at the poster

Progress since last week:

- Finished the project description part of the requirements document

This week we were still working on the requirement document. We were a bit behind schedule and need to finish the requirement document as soon as possible in order to get feedback from our client and get it signed before Thanksgiving.

Friday, November 28, 2014

Fall 2014, Week 9:

Plans for coming week:

- Finish the poster
- Start working on the preliminary design document

Progress since last week:

- Requirements document finished and submitted
- Requirements document uploaded to the project site
- Start working on the poster

Finally we finished the requirements document and got it signed. The requirements document was also uploaded to the project site. We felt like we have a problem on our time management. To prevent similar thing happens, we need to check each other's schedule and have more time to work together.

Friday, December 5, 2014  
Fall 2014, Week 10:

Plans for coming week:

Finish the preliminary design document  
Finish the progress report  
Meet with Professor McGrath for end-of-term meeting

Progress since last week:

- Preliminary poster finished
- Preliminary poster uploaded to the project site

This week we started working on the poster that we will be using in the Expo and got it finished in this week. Since this is only a preliminary design of the poster, we did not fill in all the contents. The poster have a blue background and orange header and it follows the Firefox OS branding guidelines.

Two of our member, John and Jonathan, went to Portland to meet with our client and engineers from Mozilla. We have a more clear direction on what we are going to do.

Friday, December 12, 2014  
Fall 2014, Finals week:

Plans for the next term:  
Working on the Javascript prototype  
Research on Services Worker Research on RequestSync

Progress since last week:

- Preliminary design document finished
- Progress report finished

This term we spent most of the time working on documents and research. Next term we will start working on the prototype of our API. and we will later move on to the implementation in C++.

This week is the end of Fall term. The blog will stop updating until next term.

Friday, January 9, 2015

â€¢ Winter 2015, Week 1:

Plans for coming week:

- Start working on the actual implementation

Progress since last week:

- Set up the weekly meeting with our TA
- Set up our group meeting time
- Requirements updated

Winter term begins and we get back to our project. In the first week, we scheduled a weekly meeting with our TA. and updated the requirements. We broke down our project into 24 requirements and each member handle 8 of them. We are expected to complete 6 requirements at the end of this term and 2 for the next term.

Friday, January 16, 2015

â€¢ Winter 2015, Week 2:

Plans for coming week:

- Start working on the prototype API

Progress since last week:

- Updated requirements approved
- Requirements completion list submitted

This week we got our new requirements approved and divided works to each member. We also started our implementation of prototype API using JavaScript.

Friday, January 23, 2015

Winter 2015, Week 3:

Plans for coming week:

- Working on the prototype API

Progress since last week:

- Updated requirements approved

Again we update one of the requirement to make it more specific and got it approved. After speaking with our sponsor, we decided to implement our prototype API as a Firefox addon instead of putting it into the Firefox core, while the C++ API will still need to be put into the core.

Friday, January 30, 2015  
Winter 2015, Week 4:

Plans for coming week:

- Finish the implementation of the prototype API
- Finish the first 9 requirements and get ready for demo

Progress since last week:

- Created a draft of the prototype API

This week we were working on the prototype API. Because our API will communicate between C++ and Javascript, we implement our prototype API as a XPCOM component. We successfully installed the prototype API as an addon to the Firefox but we failed to call the C++ function in Javascript. We will keep working on it and try to make the addon works in the next week.

Friday, February 6, 2015  
Winter 2015, Week 5:

Plans for coming week:

- Finish the implementation of the prototype API
- Test the prototype API on desktop, Android and Firefox OS
- Demo the prototype API to the TA

Progress since last week:

- Implemented the prototype API addon using addon SDK

- Battery level, device charging state and network latency data are accessible from the prototype API

This week we were still working on the prototype API. Since the addon-via-manifest prototype API does not work well, we tried another method and implemented the prototype via SDK. Also, it turns out that some of the functionality of our API are only available on the Firefox OS but not the desktop version. Therefore we need to write a new prototype for the Firefox OS device.

Friday, February 13, 2015

Winter 2015, Week 6:

Plans for coming week:

- Start working on the actual API using C++ or Javascript

Progress since last week:

- Prototype API finished
- Completed the demo of first 9 requirements to the TA

This week we finally finished the prototype API. The prototype can detect the battery level, charging state of the device, network connection type and shows latency-related information. We also changed our requirements after talking to the sponsor to scope the prototype down so that we have more time to focus on the actual API.

Friday, February 20, 2015

Winter 2015, Week 7:

Plans for coming week:

- Continue working on the actual API using C++ or Javascript

Progress since last week:

- Start working on the actual API using C++ or Javascript

This week we keep on working on the API. A few weeks before we got a developer reference phone called Flame from our sponsor. We began testing our API on this Firefox OS device to make sure our API works on all platform of Firefox.

Friday, February 27, 2015  
Winter 2015, Week 8:

Plans for coming week:

- Continue working on the API using C++ or Javascript

Progress since last week:

- Continue working on the API using C++ or Javascript

This week we are still working on the API. Most of the functions of the actual API are same as the prototype API. Some of the new functions of the actual API include collect network status information, access system load information and to integrate with existing API such as ServiceWorker and RequestSync. Each of our group member take charge of one of these function.

Friday, March 6, 2015

Winter 2015, Week 9:

Plans for coming week:

- Finish the next 9 requirements
- Finish the implementation of the API

Progress since last week:

- Continue working on the implementation of the API

We are going to demo the next 9 requirements to the TA in the final week. We hope to finish the implementation of our API next week and test it before we demo.

Friday, March 13, 2015

Winter 2015, Week 10:

Plans for coming week:

- Finish the next 9 requirements
- Finish the implementation of the API
- Demo the 9 requirements to the TA

Progress since last week:

- Poster updated
- Continue working on the implementation of the API

This week we updated the poster and keep working on the API. We are going to demo the requirements next week but some of the requirement are still incomplete. Some of the functions of our API took longer than we expected to implemented.

Friday, March 20, 2015

Winter 2015, Finals week:

Plans of the next term:

- Finish the implementation of the API
- Prepare for Expo

Progress since last week:

- Finished the demo of winter term requirements

This term we finished most of the implementation of the API. We tested our API mainly on Firefox desktop. In the next term we will focus on the implementation and testing of our API on Firefox OS.

Friday, April 3, 2015

Spring 2015, Week 1:

Plans for coming week:

- Continue to work on the FxOSUService API

Progress since last week:

- Set up the weekly meeting with our TA

Spring term begins. This week we scheduled a weekly meeting with our TA. We have to complete the last 6 requirements before the Expo and prepare for the Expo on May 15th.

Friday, April 10, 2015

Spring 2015, Week 2:

Plans for coming week:

- Continue to work on the FxOSUService API

Progress since last week:

- Met with our sponsor via video call

This week we had a meeting with our sponsor. We keep on working on our API and hope to finish it quickly so that we have time to work on the test app.

Friday, April 17, 2015

Spring 2015, Week 3:

Plans for coming week:

- Continue working on the FxOSUService API
- Working on the test app

Progress since last week:

- Updated the poster

This week we are still working on the requirements. We also start working on a test app that benchmark the performance of our API. The test app is implemented in Javascript. We also updated our poster.

Friday, April 24, 2015

Spring 2015, Week 4:

Plans for coming week:

- Continue working on the FxOSUService API
- Continue Working on the test app
- Update the poster

Progress since last week:

- Modify the format of our poster

This week we need to modify the poster to a specified format. It won't be hard because we only need to change the layout and content can stay the same. We need to finish our poster before May 1st. We are also working on the final draft of our API and hope to get it done by next week.

Friday, May 1, 2015  
Spring 2015, Week 5:

Plans for coming week:

- Finish the test app
- Demo the requirements to our TA

Progress since last week:

- Finished the poster
- Finished our API

This week we finished the poster. The final draft of our FxOSUService API is finished and is working. Expo is approaching. We are going to demo our requirements to the TA and prepare for the Expo after the demo.

Friday, May 8, 2015  
Spring 2015, Week 6:

Plans for coming week:

- Prepare for the Expo

Progress since last week:

- Finished all the requirements
- Finished the test app
- Finished the demo

This week we finished the test app and finished the demo of our requirements. Expo is next Friday and we have to start prepare for it.

Friday, May 15, 2015  
Spring 2015, Week 7:

Plans for coming week:

- Start working on the report and presentation

Progress since last week:

- Expo is over

Expo is over.

## 4 Project Poster

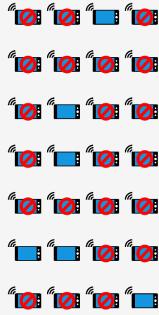
This is the final project poster that we used when demoing our project at the Engineering Expo.



## Firefox OS

### Introduction and Background

Firefox OS is an open source operating system for smartphones developed by Mozilla. It is a web-centric system based on a Linux kernel. Firefox OS phones have launched in 15 countries around the world, and targets markets in developing countries.



### Problems

- Unreliable Networks
- Unreliable Electricity
- Intensifiers
- Inefficient Data Usage
- Overpopulation
- Exponential Increase in Number of Users

### Project Description

Developers need a way of knowing and adjusting their applications behavior depending on the condition of the network. Our solution was an API that monitors various conditional data and uses the information to determine if a network request should be attempted. This data includes battery information, network statistics, memory usage and charging status.

This API lives in the network stack (Necko) of the web runtime layer called Gecko. It is written in Javascript and is callable by Javascript running in a web context.

## NETWORK RELIABILITY API FOR FIREFOX OS

### Improving efficiency through analysis of network metadata

#### HOW FIREFOX OS CONNECTS HTML5 TO HARDWARE



### Importance

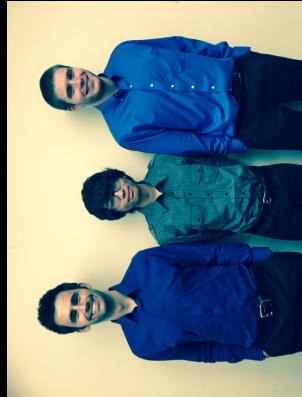
This project promises to improve efficiency for millions of people around the world using Firefox OS. Additionally, the web IDL specifications we have drafted will serve as a blueprint for an open web standard, determining how to handle this problem on all devices, keeping the web open for all.

### Conclusions

**Saying yes is always better.** Our goal is to improve the user experience. When insufficient data is available we should encourage a request to be attempted. Blocking a request based on insufficient data would degrade user experience, and the efficiency gains made possible with this API.

**Work yet to be done.** Gecko does not yet have APIs that access to all of the information which could be useful for this API. There is currently no API which fetches data regarding CPU usage nor is there one which relays how much memory the host device has available. This makes monitoring system load a challenge.

Our Team: FxOSU



### Team Members

- John Zeller,  
[zellerj@onid.oregonstate.edu](mailto:zellerj@onid.oregonstate.edu)
- Pok Yan Tjam,  
[tjampe@onid.oregonstate.edu](mailto:tjampe@onid.oregonstate.edu)
- Jonathan Mcneil,  
[mcneili@onid.oregonstate.edu](mailto:mcneili@onid.oregonstate.edu)

### Client

- Dietrich Ayala,  
[dietrich@mozilla.com](mailto:dietrich@mozilla.com)
- Thanks to our client and all of the Mozilla community that helped us with this project!



### Results

1. Not all available data is always useful.
  - For Example, Memory Usage alone
  - Certain data is intentionally unavailable
    - Network Type (Wifi, USB, 4G, etc.)
  - Certain information is only available on certain platforms.
  - Network Statistics APIs only available on Firefox OS.
2. Certain data is intentionally unavailable
  - Network Type (Wifi, USB, 4G, etc.)
3. Certain information is only available on certain platforms.

**Oregon State**  
UNIVERSITY

## 5 Project Documentation

1. How does your project work?

What is its structure?

What is its Theory of Operation?

Block and flow diagrams are good here.

2. How does one install your software, if any?
3. How does one run it?
4. Are there any special hardware, OS, or runtime requirements to run your software?
5. Any user guides, API documentation, etc.

## 6 Learning New Technology

1. What web sites were helpful? (Listed in order of helpfulness.)
2. What, if any, reference books really helped?
3. Were there any people on campus that were really helpful?

## 7 What did we learn from all of this?

### 7.1 John Zeller

1. What technical information did you learn?
2. What non-technical information did you learn?
3. What have you learned about project work?
4. What have you learned about project management?
5. What have you learned about working in teams?
6. If you could do it all over, what would you do differently?

## **7.2 Pok Yan Tjiam**

1. What technical information did you learn?
2. What non-technical information did you learn?
3. What have you learned about project work?
4. What have you learned about project management?
5. What have you learned about working in teams?
6. If you could do it all over, what would you do differently?

## **7.3 Jonathan McNeil**

1. What technical information did you learn?
2. What non-technical information did you learn?
3. What have you learned about project work?
4. What have you learned about project management?
5. What have you learned about working in teams?
6. If you could do it all over, what would you do differently?

## **8 Appendix I**

1. Essential Code Listings. You don't have to include absolutely everything, but if someone wants to understand your project, there should be enough here to learn from. If you worked within a larger project, something like a patch file might be a good way to go.

## **9 Appendix II**

1. Anything else you want to include. Photos, etc.