NETWORK RELIABILITY API FOR FIREFOX OS
Improving Efficiency Through Analysis of Network Metadata

May 22, 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 form 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:

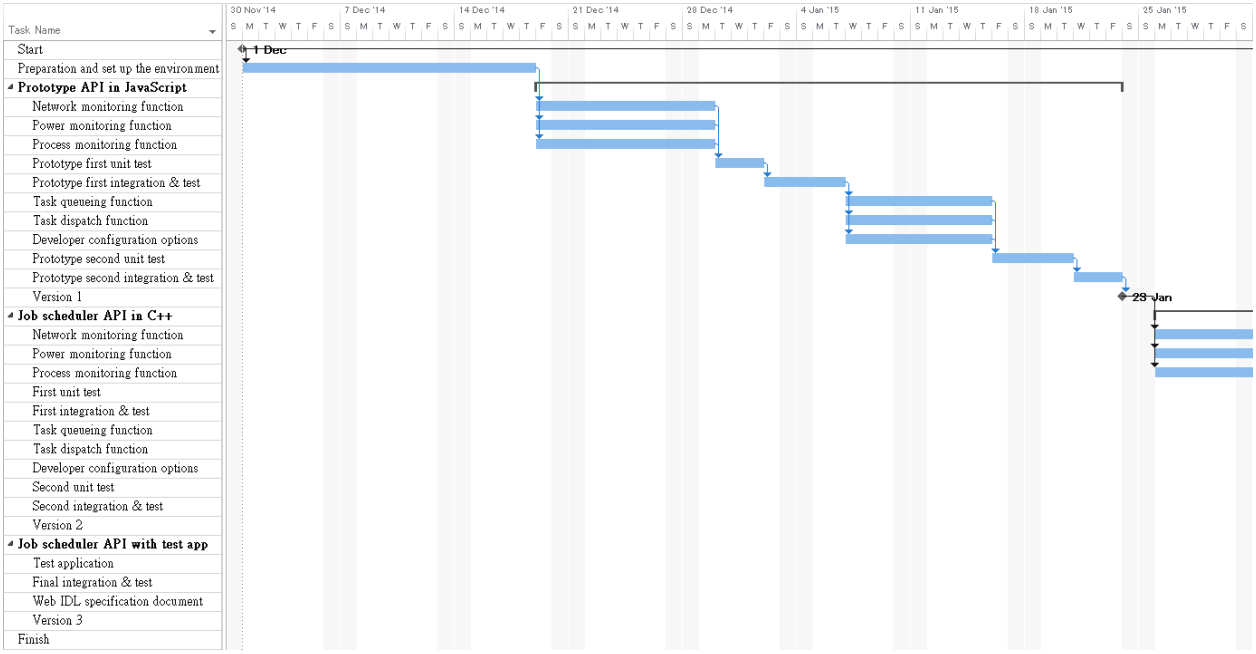
One of the challenges associated with creating an Programming Interface that is designed to minimize power usage is creating the interface in a way that doesn't hurt more than it helps. It could be useful if we could create a functionality that can schedule tasks, but if our method of doing so costs 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.

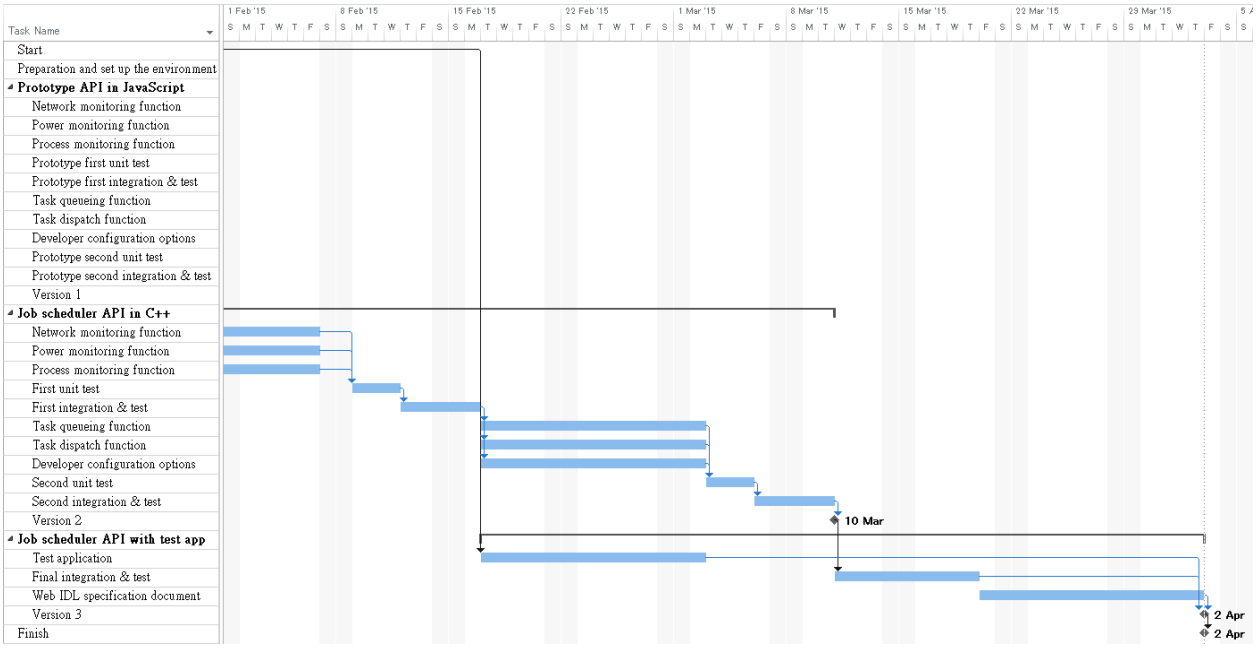
For any of the challenges associated with this project, both foreseen and unforeseen, there are a couple of options for what to do. Development in the mozilla core is not something that is being done alone. There is a thriving IRC network with developers with far more experience than any of us. The first 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

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 Tjiam	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 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 protoype 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 force 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.

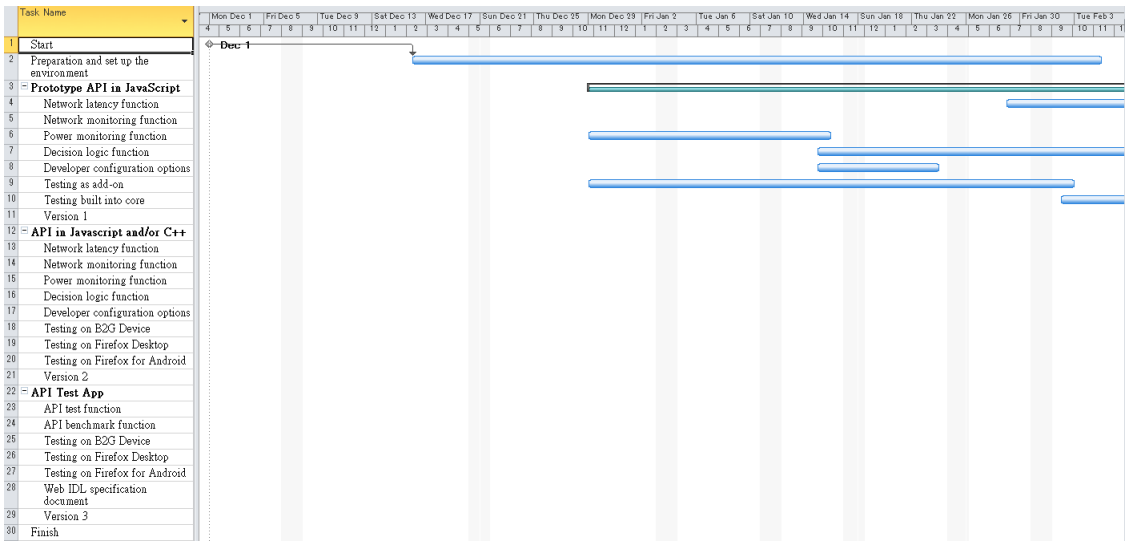


Figure 1: 1 of 3: The final Gaant chart for how our project timeline actually turned out.

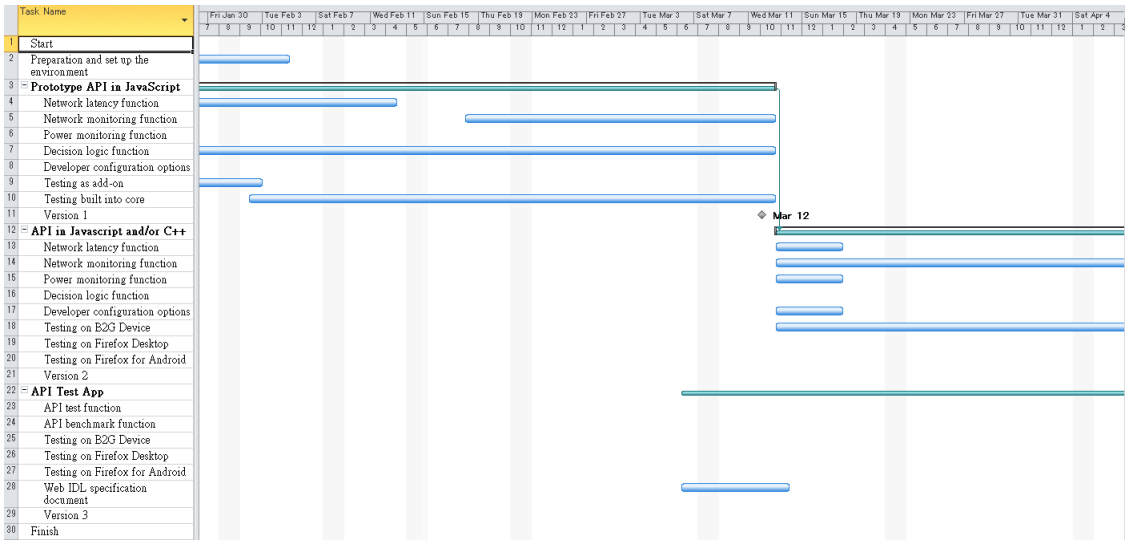


Figure 2: 2 of 3: The final Gaant chart for how our project timeline actually turned out.

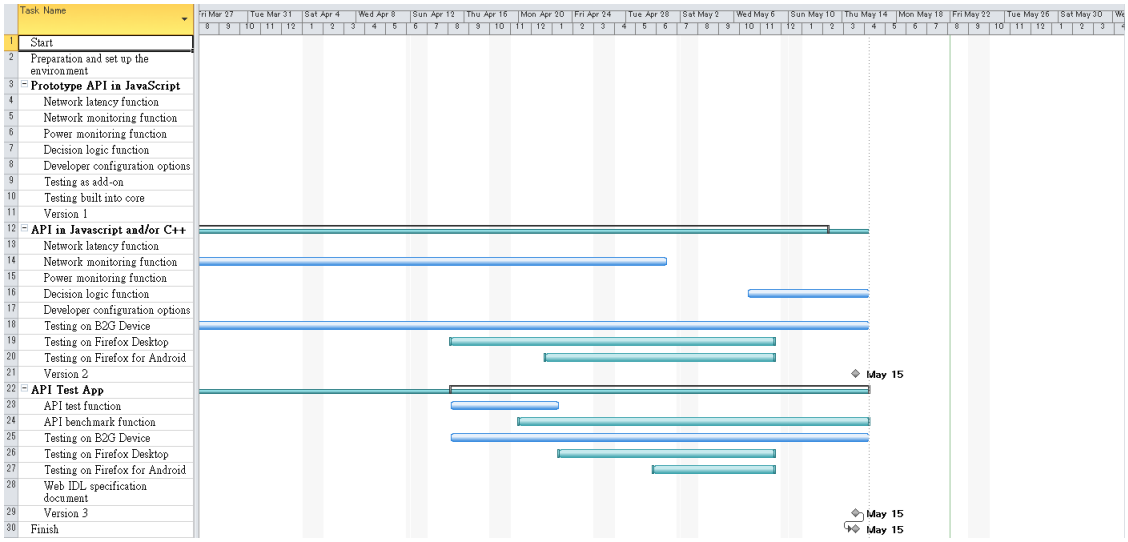


Figure 3: 3 of 3: The final Gaant chart for how our project timeline actually turned out.

3 Weekly Team Blog Posts

1. These should be formatted nicely and clearly distinct from one another.

4 Project Poster

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

NETWORK RELIABILITY API FOR FIREFOX OS

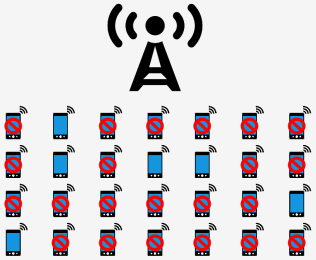
Improving efficiency through analysis of network metadata

mozilla

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.

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.

Steps and Challenges

Data Collection:
- What data is available?
- How do we obtain it?

Data Analysis:
- Is this data useful?
- How can we use it to determine if it is a good time to make a network request?

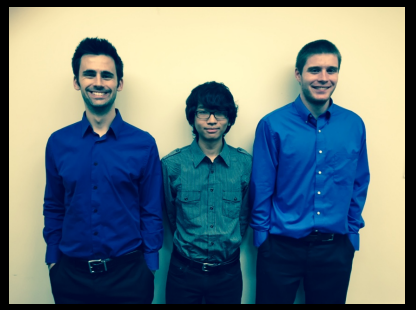
Cross-Platform:
- What data is available on this platform?
- Do we know what is and isn't available?

Exposing to Developers:
- How do we make this available to developers?
- Where in the Firefox code base should this live?

Results

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

Our Team: FxOSU



Team Members

- John Zeller, zellerjo@onid.oregonstate.edu
- Pok Yan Tjiam, tjiamp@onid.oregonstate.edu
- Jonathan Mcneil, mcneilj@onid.oregonstate.edu

Client

- Dietrich Ayala, dietrich@mozilla.com

Thanks to our client and all of the Mozilla community that helped us with this project!

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.