

Pre-Project Bachelor Report

Project ID: HPR/D-2015/015

Written By: Per-Arne Andersen, Paul Richard Lilleland

Finalized Date: 04.03.15

Gruppe 15

Forprosjekt Bacheloroppgave 2015

**Global Online Traffic, Hacking and Monitoring
(GOTHAM)**

Per-Arne Andersen, Paul Richard Lilleland
04.03.2015

Supervisors

Christian Auby, Sigurd Brinch

Table of Contents

1. Project description	4
1.1 General information	5
1.2 Background	5
1.3 Goal	5
1.4 Outcome	6
1.5 Purpose	6
1.6 Organization	7
1.7 Code standard	7
2. The project	8
2.1 Goals	8
2.2 Limitations	9
2.3 Prerequisites	9
2.4 Phases	11
3. Solution Alternatives	12
3.1 back-end server	12
3.2 Front-end client	13
3.3 Graphics Library	14
3.4 Other tools	16
4. Tech design	17
4.1 back-end (server):	17
4.2 Front-end (client):	20
5. Game Design	22
5.1 Overview	22
5.2 Menu	22
5.3 Gameplay	23
5.4 Ethics	23
6. Future Work	24
7. Progress Ferdigstill Gantt	25
8. Appendix	26
8.1 Code standard list	26
8.2 Traffic structure diagram	27
8.3. Group contract	28
8.4 References	29

1. Project description

Gotham (Global Online Traffic Hacking and Monitoring) is a simulator of how the Internet infrastructure and the different layers of the of the OSI model work.

Internet traffic should be represented by different environment variables such as Streaming, file transfer and other protocols. The infrastructure consists of 3 tiers, Tier 1 being the “highway” and Tier 3 the end-user ISP. To be able to simulate this amount of traffic without a supercomputer, the traffic between the nodes will be generated by the server or from statistics on the Internet.

To achieve the detailed traffic information required to get a peek at the lower layers of the traffic, a micro traffic session is queried and initialized if this should be requested. This is a traffic session between for example an end-user and a video-streaming site. With the stream open, the student will be able to see the traffics physical path through the world along with the packet headers on the different layers.

Traffic paths is generated by algorithms like A*, Dijkstra or ACO (Ant Colony Optimization). With a little information about the different algorithms the student should be able to see examples of the different algorithms and how they work.

The front-end is highly likely to be programmed in JavaScript. This should feature a dashboard with a world map for a good overview for the student with needed tools and functionality available from a side menu. In order to make the product more appealing, the application will be gamified. This will feature objectives the user can do inside the dashboard.

This would be presented in the form of a hacking game.

Every network operator should be aware the risk of their system being penetrated, this will feature some common issues such as old version exploits, weak passwords and more. This will hopefully give an insight of the seriousness and importance of keeping a system as impenetrable as possible.

1.1 General information

- **Title:** Global Online Traffic, Hacking and Monitoring (GOTHAM)
- **Project ID:** HPR/D-2015/015
- **Participants:** Per-Arne Andersen, Paul Richard Lilleland
- **Course:** Data
- **Date:** 04.03.15
- **Contractors:** Self-Composed
- **Supervisors:** Christian Auby, Sigurd Brinch

1.2 Background

To achieve a bachelor degree in Computer Engineering at the University of Agder, the students are required to complete a 20 credits assignment during the sixth semester of their study program. The assignment was composed and chosen by the students themselves and approved by the university. The task is inspired by a game called Uplink, where your objective is to use your tools, hardware and skills as a hacker to complete different assignments.

There are multiple reasons why this task was chosen. Primarily, experience shows that the end result is proportional with how interesting the assignment is for the specified group. The task requires good coding skills, the development and implementation of algorithms and a good amount of background information on how the world IT infrastructure works. Due to the fact that this is a learning tool for network and hacking, the gamification on top of the simulator makes the project even more compelling.

1.3 Goal

The goal of the project is to give IT students a tool for understanding how the Internet infrastructure and traffic works. The traffic containing protocols, services, packets and the different network layers can be difficult to understand. This tool will hopefully give a better overview of how the different components are connected. The application will also have a gamified aspect containing ways of manipulating the traffic. This could for example be a denial of service attack on a specific node to achieve an alternative routing of the traffic going through it or a man in the middle attack to obtain sensitive data.

1.4 Outcome

To use the knowledge, experience and skills achieved during the study period to develop an application. With the groundwork and report writing needed in an enterprise environment together with a cooperative team this hopefully gives a result that meet the requirement for a good grade.

1.5 Purpose

Hacking exist in a lot of games, a few games are based purely on the life of a “Hollywood hacker”. Examples of this would be a game series called Hacker or the more modern and popular game called Uplink.

Even though these games are entertaining, none of them are based on a realistic infrastructure. The goal of this project is to develop a product which can teach a user how the Internet Infrastructure in the real world works.

The hacking part of the product will be simplified as the group members only have a general course in ethical hacking, and the implementation would require a massive library of tools to achieve a realistic experience.

Some enterprise simulators of large scale networks exists but this is mainly for testing and benchmarking purposes. Gotham’s focus is on teaching the fundamentals of the world’s IT infrastructure through the entertainment of hacking.

1.6 Organization

Division of labor

As the group only consists of two members, no detailed division of labor seems necessary as the members mostly are in contact while working. If a team member have issues finding a good solution, both team members can cooperate to get a better result.

The project is split into two parts, one for the back-end server, and another for the front-end game client. Paul Richard Lilleland will be in charge of the back-end server and Per-Arne Andersen of the front-end client.

Each member of the team is responsible that his part of the code follows code standards and is compilable.

As both members of the team have a consistent workflow the distribution evens itself out.

Meetings

Each Tuesday at 13:00 the group meets with the main supervisor to give an update on the project and to get feedback on their work, what they should prioritize and technical help.

Two additional supervisors have agreed to work with the group each having different areas of expertise relevant to the project. Meeting with the other supervisors is scheduled when needed.

The group are responsible for documenting each meeting with a simple report.

1.7 Code standard

The group members have agreed on a list of code standards to keep the code structured and consistent through the project period.

A list of requirements may be found in chapter 8.1

This is generally what is described in the C# coding convention guide from Microsoft found here:
<http://msdn.microsoft.com/en-us/library/ff926074.aspx>

2. The project

2.1 Goals

A: Internet Simulation

- Node Generation (Routers)
- Traffic Simulation between Nodes
- Front-end including a basic world map with traffic information
- Data (nodes, cables, people, ISPs)

B: Detailed Node Simulation

- Extract and present real traffic data from the Internet
- Import Real World ISP Names
- Algorithms (Traffic routes, Node generation, Traffic statistics calculation and more)
- Implement simulation of the most relevant protocols and services
- Gamification by developing a simple objectives system.
- Game objects like money, hardware and software.
- Create Internet site (<http://gotham.no>)

C: Interaction with Traffic and Gamification

- Develop a "Creative Mode". This mode will allow users to interact with traffic flow by for example cutting cables between nodes.
- Simulate micro traffic on a package level (Header information)
- Hacking operating system (Incorporated into existing GUI)

D: Nice to Have

- Advanced and Demanding missions
- Rivals (Other hackers and official crime fighters)
- Time of day affects traffic
- Detailed traffic flow between nodes (Would probably require very good algorithms)
- Botnet

The goals in category A are primary goals and do in turn contain a list of goals prioritized according to necessity. A being the vital parts, B should be included. C is secondary features we hope to implement and D is room for extension. It is not planned to implement features in the D category unless there is a lot of extra time.

2.2 Limitations

The project has nearly unlimited possibilities when it comes to development and expanding. This makes it important to narrow down the goal for this project period. Focusing on the basic principles of the simulation will be the main priority along with developing an interactive interface.

Because of the complexity of the task, simpler algorithms and models will be implemented first, and expanded later during the project period.

This makes it important to plan and develop a good core structure from the start to prevent restructuring.

2.3 Prerequisites

The group have agreed to use the following tools, services and libraries in the project. There may be changes along the project period, and it is expected that additional tools is used throughout the period. Unlike chapter 3 (Solution alternatives), these meets the requirements and does not require prototyping and alternatives.

Development:

Visual Studio 2013 is the leading IDE for development in .NET. This is officially supported by Microsoft and features the required tools and structure for the development of this project.

Navicat Premium is a database management tool that supports numerous types of databases with a user friendly interface.

SourceTree is Atlassian's Git management tool with a user interface. The alternative would be Git Bash but since all group members is familiar with SourceTree this is probably a more effective tool.

Chrome is one of the leading Internet browsers. This features a great tool for debugging JavaScript code which is used in the front-end.

Services:

MariaDB is derived from MySQL and doesn't provide much difference in this case. The main reason for using MariaDB is for supporting the developer.

nginx is a fairly new web server which is excellent for fast and static content with low resource requirement.

Google Maps is the leading global map providing service with an excellent API for gathering geo information.

Libraries:

nHibernate is the .NET equivalent of Hibernate which is a library for database management in Java. nHibernate features great support for communication and To/From database object conversion.

Pixi.js is a low level, high performance graphics library for JavaScript. More about this in chapter 3.3

JSON stands for "*JavaScript Object Notation*" and is an easy to read simplistic structure for data transfer. More about this in chapter 3.4

log4net features a framework for structured logging in a .NET application.

2.4 Phases

The estimated time requirement for each phase is described in the Gantt diagram in chapter 7. This chapter features a short description for each main phase.

Research

This is the research and prototyping phase where programming languages, libraries and tools are decided. Some of these are decided without the need of prototyping but others like front-end, graphics, and communication libraries requires testing to see if they meet the requirements.

Development

The project consists of two main development zones, front-end and back-end. Starting with back-end, a database of nodes, cables, persons and other data is generated with the possibility to communicate with the front-end. The front-end consists of a structured overview, menus data presentation and other graphical elements.

Improvements

After the main features are implemented and working, the phase of improvement and polishing starts. This phase depends on the time required to implement the main features. Elements of this phase is typically algorithms, process and graphical optimization (both visual and performance).

Report and Presentation

Maybe the most important part of the project is the end report. When the project is nearing its end the group will start writing a report which is due 27th of May. This should feature everything from testing to final features with needed diagrams and images.

About a week later (no exact date given) the group have to present their product for the supervisors. This require a PowerPoint or another similar presentation.

3. Solution Alternatives

3.1 Back-end server

Back-end is needed to run pathfinding algorithms, connection to database and general package management. This requires a language that is fast, has good libraries for database management and require a consistent structure. It is likely that the language of choice will be C#. This is because the group have the least knowledge of this language, and want to learn more about its libraries (.NET).

C#

This language is the flagship of the .NET platform developing languages. Until recently, this language was closed source, but is now open for all who wish to partake in its development. This language will be considered as the game client platform because of its strong tools, and that it has been decided that the back-end application will be running in .NET. Using the same platform makes it easier to implement network communication between back-end and front-end.

It is considerably faster than JavaScript, which makes it a strong candidate. However, C# is not well suited for the web browser.

Java

Java is a programming language developed by Oracle Corporation. For this project, Java is equally preferred with C#. Java is a fast, multi-platform language with good scalability in business applications. Java can provide all necessary tools and library for this project.

Python

Python is a multi-paradigm language excellent at calculating large amounts of data, which is why this was considered as a back-end server. However, Python is not object oriented or strong typed which prevents the ensuring of a consistent structure.

3.2 Front-end client

The main requirement of the front-end client is flexibility, available graphics libraries and multi-platform support.

JavaScript / CoffeeScript

A dynamic weak typed language with a large community. The language have good documentation and gives possibilities for a game client in any web browser. Easy to learn and gives good productivity as is a simple scripting language.

The language is however not as fast as C#, this makes it poor for heavy computations. Also it is easier to produce poor quality code because of its weak typed design. Using JavaScript will require additional focus on application design and code style.

CoffeeScript is a language which compiles down to JavaScript. It does not give any noticeable performance gains, but cleans up the JavaScript syntax. In Coffee Script, prototyping is a class and scopes are defined by tabs instead of curly brackets.

It is our belief that there is much to gain on using CoffeeScript, in terms of code structure and readability. The downside of CoffeeScript is that it is a compiled language, which means it takes longer to test new revisions.

C#

There were uncertainty of whether to use C# or JavaScript in the front-end. One great advantage would be the cooperation with the back-end as this also runs C#. Even though it is an excellent language for Windows clients, the flexibility, browser support and efficiency of JavaScript defeated this option.

3.3 Graphics Library

There are many graphics libraries for JavaScript, and selecting the best fit for GOTHAM is not an easy task. Gotham requires a library which has primitive drawing (Polygons), and the possibility to manipulate these with input events.

There are two methods which can be used in this project, being DOM manipulation or Canvas/WebGL drawing. DOM uses the CPU for processing while WebGL utilizes the GPU.

Multiple libraries have been tested in the pre project period in order to find the library that fit the project best. All of the tools used would probably have been sufficient for this project, but keeping scalability in mind, it was important to find the most stable, fast and simple library out there.

D3

D3.js [<http://d3js.org/>] is a simple DOM manipulation tool. It is not considered a graphics library, but it have all the capabilities in doing so. The D3.js team describe the library as follows;

“D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS.”

Using this library was easy, and it had features like parsing geoJSON [<http://geojson.org/>] which made it excellent for parsing geographical data. It was easy to create a detailed world map in D3.js, but this is where its greatness ends. Because D3.js is only a DOM manipulator, it hit a performance wall.

When moving the world map around, or zooming/panning it simply could not handle to translate all of DOM elements. D3.js was simply not powerful enough to render our graphical needs.

Researching further, a variation of D3.js was found, called PathGL [<http://pathgl.com/>]. This library combined D3.js and made it possible to draw inside a canvas (WebGL). This library solved the performance issues found in D3.js.

PathGL did however not have sufficient tools for our application and was therefore abandoned.

Phaser

“Phaser is a fast, free and fun open source HTML5 game framework. It usesPixi.js for WebGL and Canvas rendering across desktop and mobile web browsers. Games can be compiled to iOS and Android apps via 3rd party tools.”

Phaser is a great framework for game development, and implementing the map was an easy task. Phaser did however not fit well with the coding standards of the application. Phaser had quite a few bugs that were encountered. For instance, outline on Graphics were not correctly drawn. Phaser is currently in version 2.2.2, and development on 3.0 is currently their main focus. Our biggest concern is that Phaser 3 will not be compatible with 2.x because of its new rendering engine

[<http://www.html5gamedevs.com/topic/11783-phaser-3-development-log-we-jan-16th/?p=68025>].

Phaser is therefore not a good fit for the project.

EaselJS

EaselJS is *“A JavaScript library that makes working with the HTML5 Canvas element easy. Useful for creating games, generative art, and other highly graphical experiences.”*

It is developed by a company called CreateJS. There are multiple products available for use with the EaselJS library which makes it easier to develop advanced features swiftly.

This library did however not support extending very well, which made it hard to incorporate custom code in the library. This is required in order to get SignalR work by adding callbacks to objects in the application.

Also, some performance issues arises when attempting to translate all objects in the canvas. This is not a good sign this early on in the project regarding performance.

Pixi.js

Pixi.JS is developed by GoodBoy Digital and is described with *“Pixi.js is a devoted rendering engine. There are a host of other engines covering game, sound and physics etc. and they all work beautifully with Pixi.”*

Pixi.js is claimed to be the fastest rendering library [<http://www.pixijs.com/> | Fastest Kid in town], and combined with its sleek and easy to use API it makes a strong candidate.

One downside with Pixi.JS is that it does not bundle with Sound, Tweening, Advanced Preloader or Networking. However Pixi.JS is easy to extend so that this would not be an issue.

This library is an excellent fit for the project.

3.4 Other tools

The project requires means of communication between server and client as well as the hosting of static content (web server).

SignalR

"ASP.NET SignalR is a library for ASP.NET developers that makes it incredibly simple to add real-time web functionality to your applications. What is "real-time web" functionality? It's the ability to have your server-side code push content to the connected clients as it happens, in real-time."

This serves as the main communication library between the clients and the server.

Alternatives: *Websockets-sharp, XSockets, Alchemy Websockets, Super Websocket*

JSON

"JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate."

This is the string format of the data is transferred between client and server. And was chosen due to the readability, simplicity and support in .NET

```
{"IP-Address": "43.65.122.53", "IP-Version": "4"}
```

Alternatives: *XML*

nginx

"nginx (pronounced engine-x) is a free, open-source, high-performance HTTP server"

Due to past experience and the lightweight of the application, this was chosen as the web server.

Alternatives: *Apache, Lighttpd, IIS*

Google Maps

This is Google's official world map service. Due to the maturity and support for API, this was chosen as the geographical information service.

Alternatives: *Bing, Yahoo, WikiMapia and more*

4. Tech design

The base modules consists of a database, back-end and front-end. The database will be hosted by a MariaDB server communicating with the back-end server using the nHibernate library. The back-end server will also communicate directly with the clients connected using Newtonsoft's json.net library. The client is loaded from a webserver running nginx. After the client has loaded it starts communicating with the back-end server. Due to the necessity of quick access all necessary files is preloaded and the web server is no longer needed. Updates that requires a live feed or constant connection is established through Websockets while the rest of the data is obtained through HTTP's get and post requests.

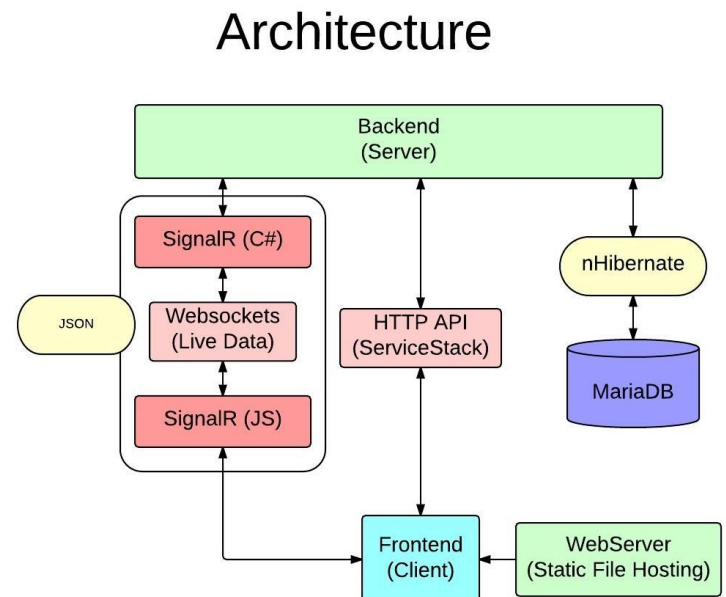


Figure 1 Architecture

4.1 Back-end (server):

The main task of the back-end is to simulate and process traffic between hosts in a virtual world. We define two types of traffic in our simulation, *Micro* and *Macro* traffic.

Macro traffic is used for statistical purposes in the simulation. This type of traffic accounts for most of the traffic. In order to get a realistic simulation, this is a requirement as it allows the application to simulate most of the network traffic with minimal overhead. Macro traffic does not contain any low level information like packet data. When detailed data simulation is required, the application generates **Micro Traffic**. This type of traffic should be recognized as how packets behave in reality. Micro Traffic contains packet information like protocol, destination and source.

This traffic requires additional processing by the server, and it is therefore important to not overuse this type of traffic in the simulation.

Another important task for the back-end server is to gather and **process data** from the Internet. If possible this will be a general load statistic of the different areas of the world. An example of this type of data would be the IPViking's (<http://map.ipviking.com>) live chart of cyber-attacks going on around the world.

The communication with the front-end happens with strings parsed to **JSON** format and transferred with SignalR. **SignalR** makes it possible to call functions on the client and vice versa, which makes it very handy, simple to implement and use.

When a micro traffic request is queried, the server builds an object called "**Connection**". This object contains a host and all communications from that specific host to any other host. A Connection contains multiple "**Sessions**", which is an established connection between two hosts (source and target). Each Session will then contain all **packages** generated between the 2 communicating hosts. This could be for example a file download consisting of 500 packages each containing the necessary information and data.

When two points needs to be connected, the server has to **find a path** between the two hosts using the cable map. The database has information on all nodes, cables and node-cable connections. This is used to find a short and reliable path for the packages to travel.

When the server processes these packages the data is transferred directly as there is no reason to use server resources to transfer them between each node. Variables like latency and reliability is calculated from the nodes and cables in the path.

Traffic Processing:

The traffic will be managed as shown below. Depending on the possibility of gathering statistical data from the Internet the data will either be downloaded or generated by server.

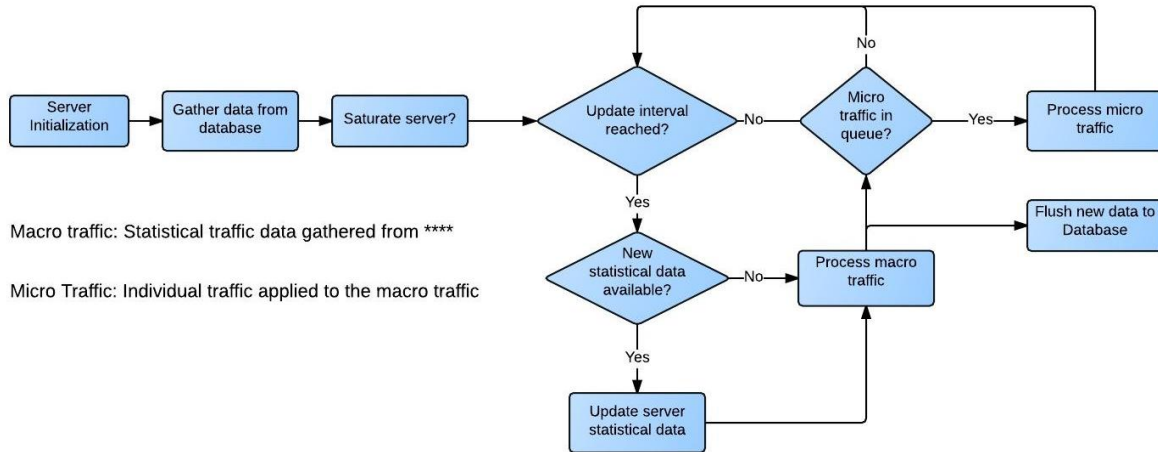


Figure 2 Traffic processing

Traffic Structure:

The micro traffic will generate traffic containing the most relevant layer of the OSI model, giving information on both layers used in the application and the layers which are not implemented. Below features the upper part of a chart of how the traffic will be structured and how it is integrated in the application. (See appendix for full diagram)

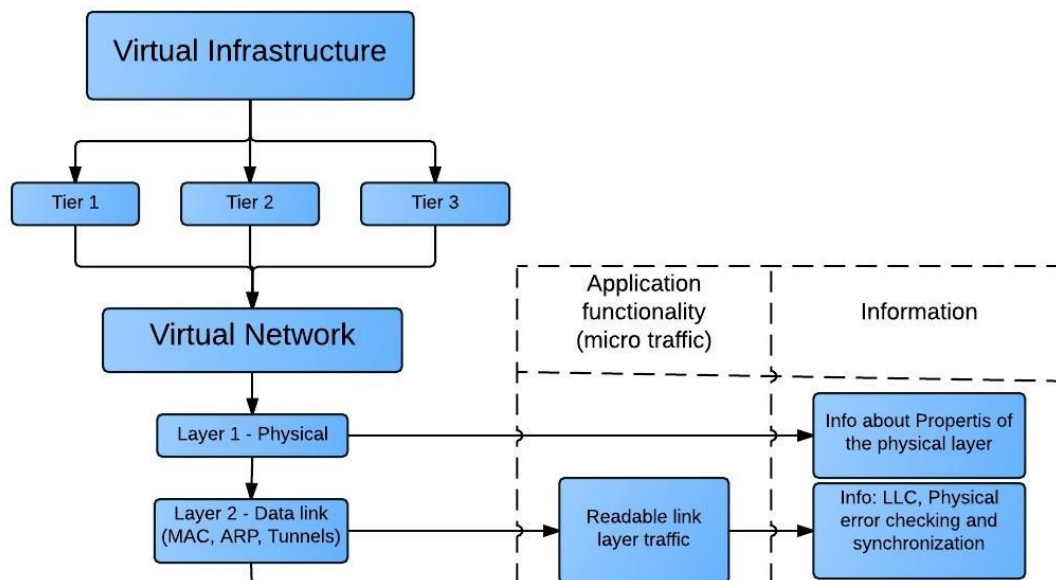


Figure 3 Traffic Structure

4.2 Front-end (client):

The front-end applications main task is to present the simulation in an intuitive way. It should be able for users to interact with the application.

Front-end features mainly a map representing the real world, this map contains Internet nodes which simulates the real Internet traffic flow.

Furthermore, it is planned for the user to have possibility to create his own network of computers to interact with the simulation in various ways. It is not yet determined how to present the interaction model, but this may for example be to manipulate the traffic capability of a node.

It is important to contribute back to the developing community, and this is done by simply splitting the framework in two; **Gotham Game Framework** [<https://github.com/perara/GOTHAM>] and **Gotham Game**.

By doing this, all generic code will be available for public use.

The Gotham Game Framework will consist of the following third party technologies (this may change in the final product):

- HowlerJS - Sound Library
- Pixi.JS - Graphics Library (Renderer)
- SignalR - Network Library

Additionally, the Game Framework will be developed in a way that it is intuitive and easy to use.

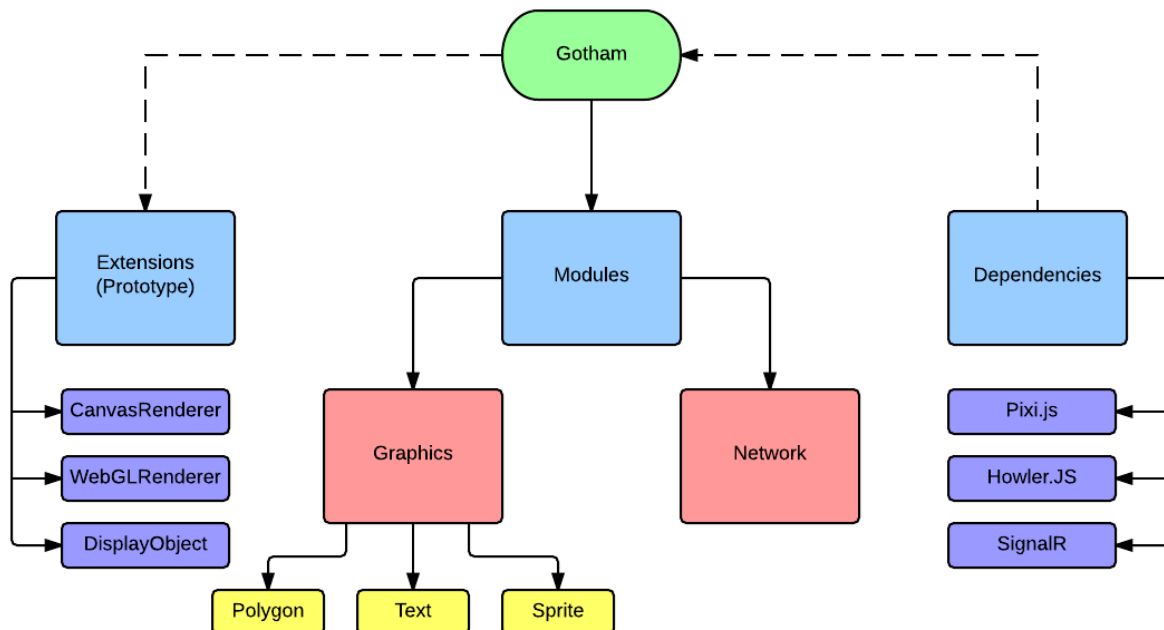


Figure 4 Front-End Structure

Our proposed design for the game framework contains mainly of modules. Each of these modules is accessed statically from the root namespace *Gotham*. This design allows for easy access to most parts of the application. For example, creating a sprite element is done by writing “*new Gotham.Graphics.Sprite*”. Most of the graphics elements inherits a Pixi.js class, for example *Gotham.Graphics.Sprite* inherits *Pixi.Sprite*. This is not always sufficient, since you cannot directly modify *Pixi.Sprite*’s parent class since JavaScript’s objects are based on prototyping, it is easy to manipulate other classes without modifying any source code.

An extension section is therefore in place to directly manipulate classes in the Pixi namespace.

The design of the game framework is complex, but using this type of structure will ease the development process of the game application.

It will also be documented so that it is possible to use it by a third party developer.

The Game Framework will be published under the MIT License.

Having a comprehensive game framework to work with, development of the game client should be fast, easy and feature an elegant code structure.

Game client will feature **scenes** and **objects**. A scene represents area of the game, for example a World Map. This scene contains multiple objects. These objects are often a collection different primitives like Rectangle, Polygon etc.

Objects are then placed inside a scene, and the scene are added to the renderer.

All code will be documented, and later parsed with a documentation tool like JSDoc or Codo. This allows for an easy to navigate web page where all documentation can be read.

5. Game Design

The front-end will probably be written in CoffeeScript, attempts have been made to create a good development flow for a JavaScript graphics renderer called PIXI.JS. This framework is built on top of canvas and WebGL.

5.1 Overview

The overview is the main screen of the game consisting of a world map and a menu bar. Additional information is likely to be implemented on the sides of the map or even a plugin function to be able to customize the screen.

Below you see an early draft of how the game client structure is planned. This will feature more nodes and cable drawings in the finished version.

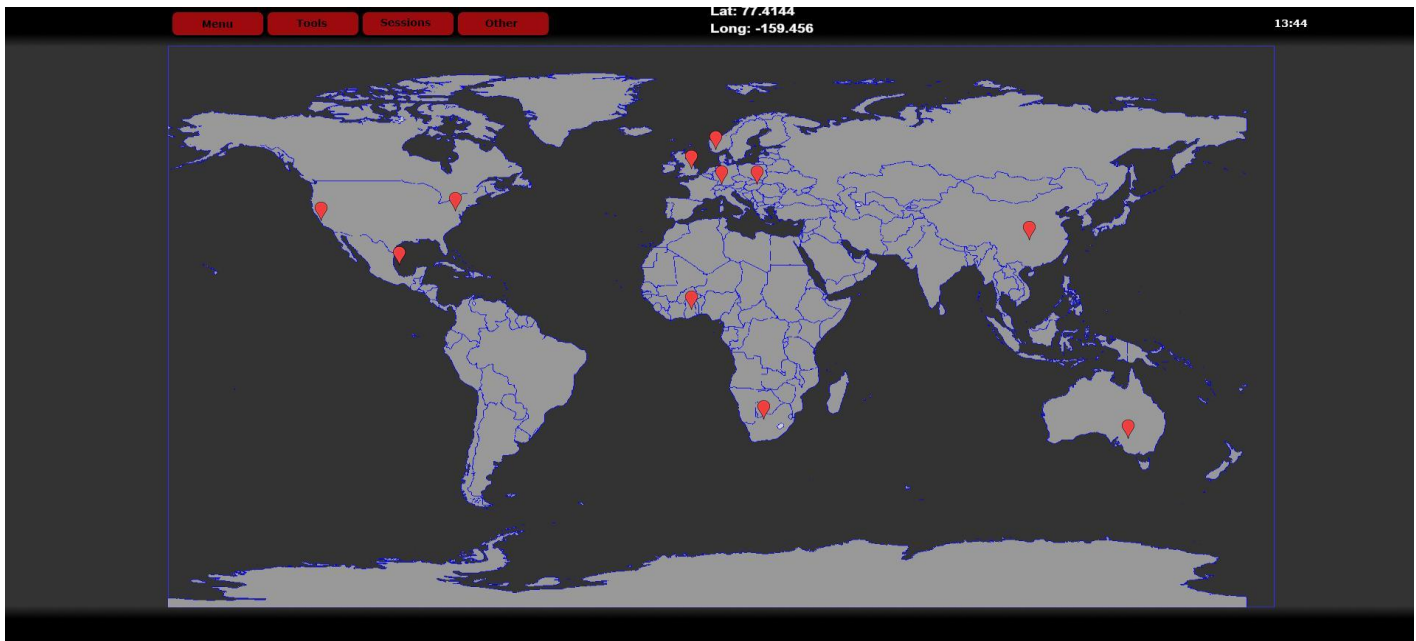


Figure 5 Client Draft

5.2 Menu

A drop down menus for features like software, hardware, locations and store. Also features information like a clock, risk level and other information.

5.3 Gameplay

Gameplay is probably the most important aspect to achieve a good experience. The game does not have any advanced graphical features and relies on good structure and features to give a good impression.

Hardware

One of the “ladder” experiences of the game will be hardware. You start with cheap low quality, low performance hardware and when you complete missions you get credits to buy better hardware which is required to accomplish better paid jobs. Examples of hardware is CPU, RAM, Transfer speed (Internet uplink), Storage.

If there is time, possibility of multiple hardware centrals will be implemented.

Objectives

To increase the fullness of the game, the possibility of doing missions and objectives will be implemented. This would be for example to intercept a traffic stream, hacking a host or running a DOS attack. If possible there will be implemented different levels of difficulty, each requiring better hardware and tools.

Console

The console will be a command line consisting of a hacking OS for running tools, checking status or other relevant actions.

Software

An essential part of the game will be software. This features tools for system penetration, monitoring and other helpful applications. If possible there will be different versions with different focus for same purpose. Like higher speed, lower quality and vice versa.

Examples of applications would be Metasploit [1] , Cain [2] or nmap [3]

5.4 Ethics

Due to the fact that the application contains unethical actions for entertainment and education in the field of ethical hacking, some information about ethics and the seriousness of illegal actions might be appropriate.

6. Future Work

This project can be expanded far past the main tasks and priorities described in this report. Some examples of future development is presented as “D” goals in chapter 2.1. Development may continue after the project period, and these goals may then become a priority in the product.

If the end result of this project prove useful for educational or entertaining purposes, some future work may be required to develop a system for maintaining the software. It is however not clear yet if this project can fit into the university curriculum.

7. Progress

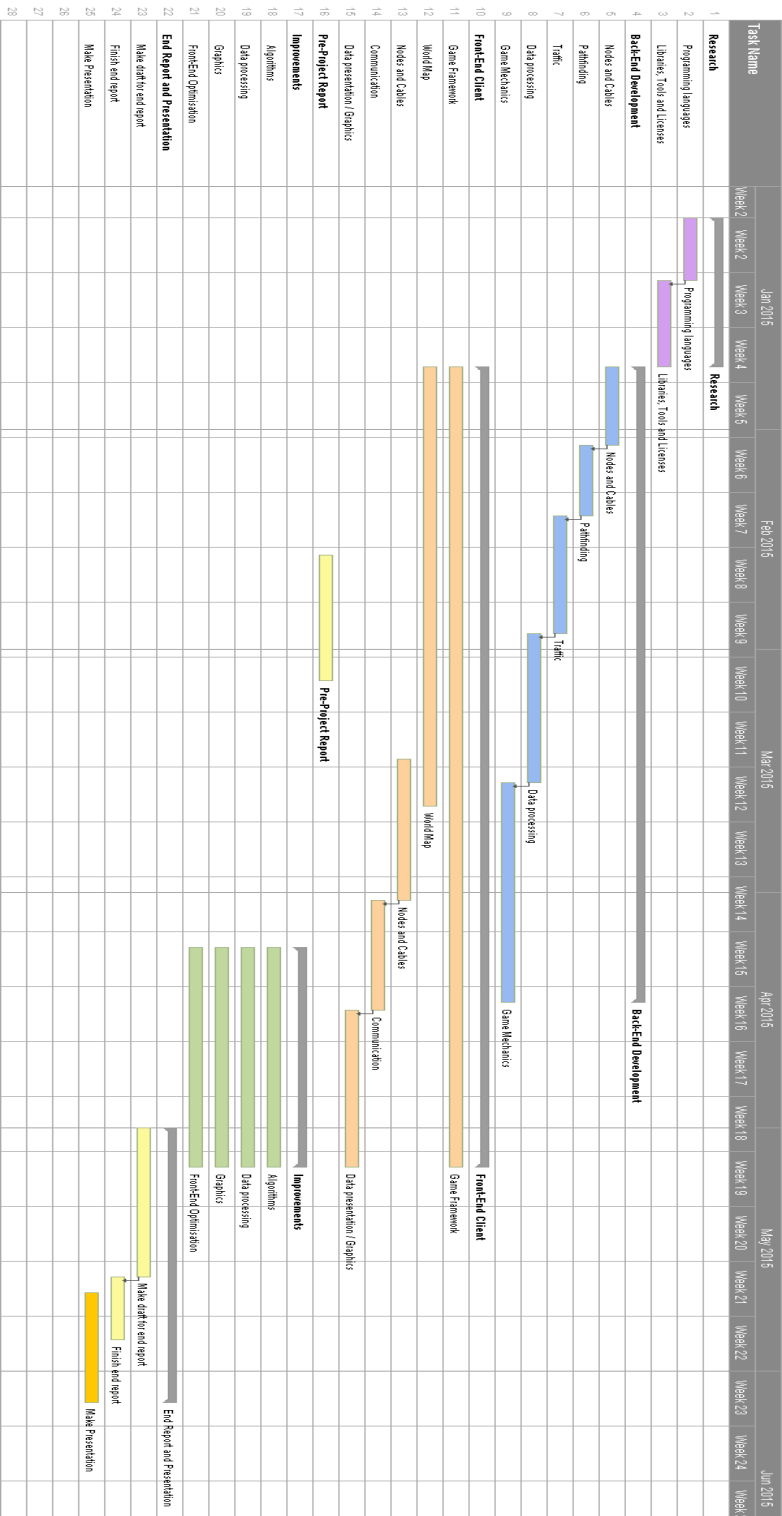


Figure 1 Gantt

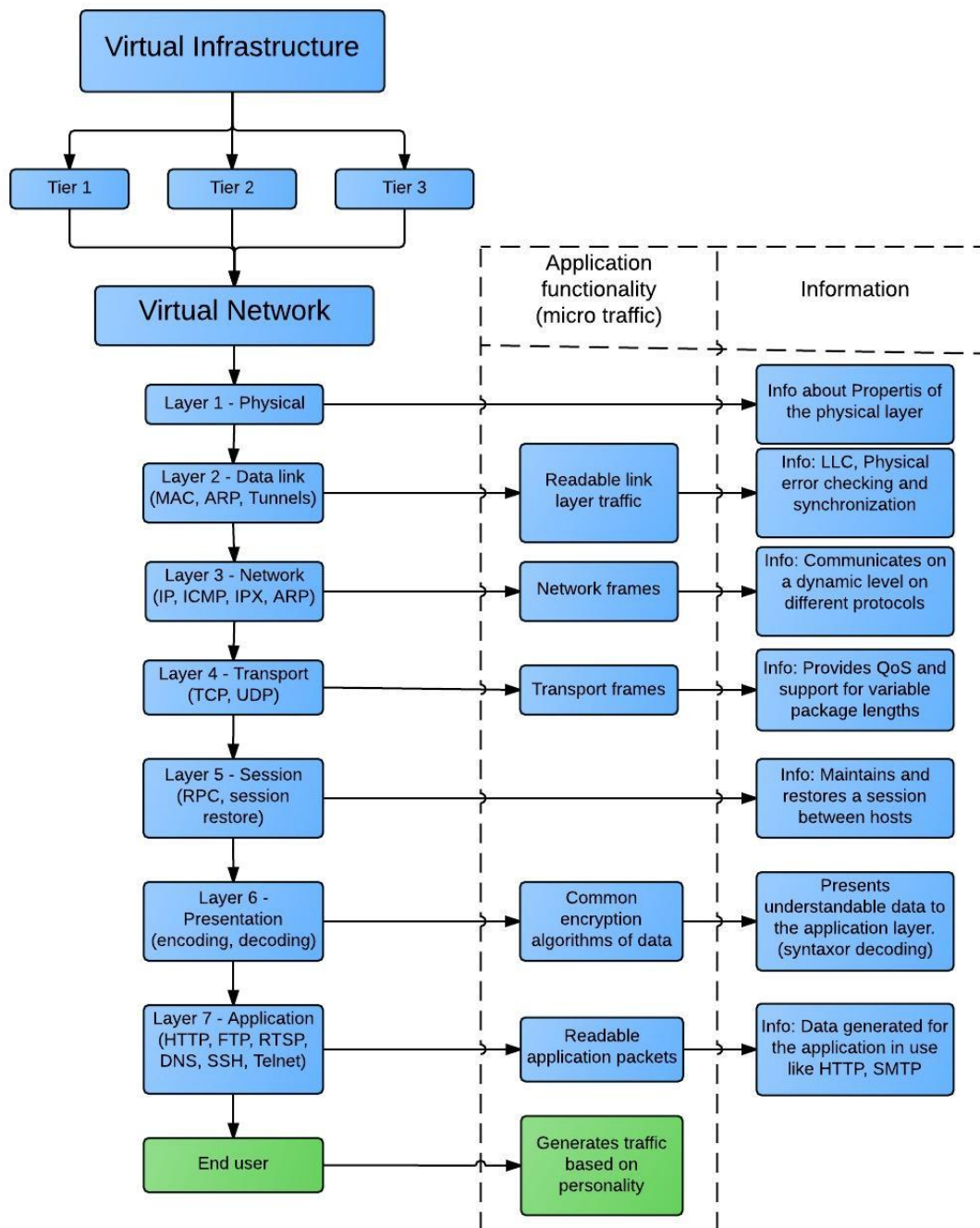
8. Appendix

8.1 Code standard list

The following programming standards and guidelines applies for the project:

- Keep the complexity as low as possible where applicable
- Code comments before the code
- Variable and function names in capital CamelCase
- “Useful” variable name wherever applicable
- Class names start with capital letters.
- Enums and static values: Capital letter only (ex. static final int TOP = 1)
- General code formatting using the tool standard (Fix indentations, remove blank lines etc.)
- Class, method or function start bracket on the next line
- Names of methods and classes should be as self-descriptive as possible
- One statement per line (not like this: a = 1; b = a*a)
- Flags are boolean only.
- Recycling: If the same “large” function is used with various variables in different places, that function has to be written as a method to limit copy-paste
- No spaces in variable and method names. If spaces have to be used in other places, “_” is used instead.

8.2 Traffic structure diagram



8.3. Group contract

Main goal:

Develop a well-tested and structured application within the project timeframe.
Deliver a product containing the specified A, B, C goals and hopefully more.
In order to fulfill this goal the group has agreed to the following

1. Participation:

It is expected that the project tasks are equally distributed and that the person tries to complete their tasks in time. If a group member can't meet at the designated time he or she is required to inform at least another group member.

2. Responsibilities:

Every group member is expected to complete their task with their best effort and ask if there are uncertainties or difficult tasks.

3. No conflict policy:

Listen and respect the other group members. Only constructive criticism. On disagreements the group member in charge of the part being disagreed on will have the final word. The other members are obligated to respect this and adapt the code to this choice by best effort.

4. Other:

1. Changes must be agreed upon by all members.
2. Short report after each meeting containing work done and work to be done.

Per-Arne Andersen

Paul Richard Lilleland

8.4 References

- [1] <https://www.python.org/>
- [2] <http://CoffeeScript.org/>
- [3] <http://d3js.org/>
- [4] <http://www.createjs.com/EaselJS>
- [5] <http://phaser.io/>
- [6] <http://www.pixijs.com/>
- [7] <http://signalr.net/>
- [8] <http://json.org>
- [9] <http://wiki.nginx.org/Main>
- [10] <https://developers.google.com/maps/web/>
- [11] <http://www.metasploit.com/>
- [12] <http://www.oxid.it/cain.html>
- [13] <http://nmap.org/>