TALLINN UNIVERSITY OF TECHNOLOGY
Facaulty of Information Technology
Department of Computer Engineering

IAG70LT

Bello Tahir Nish D 144947IASM

ENVIRONMENT FOR DATA ROW ANALYSIS

Master thesis

Supervisor: Vladimir Viies
Ph.D
Associate Professor
Consultant: Peeter Ennet
Ph.D
Researcher

Tallinn 2016

Author’s Decleration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Tahir Nish D Bello

May 26, 2016

Abstract

The arrival of the digital age has brought with it an explosive amounts of data. Today, the total amount of data which humanity is capable of storing has been reported to grow faster than the world economy. This has prompted companies and research groups worldwide to invest heavily in data analytic and big data processing. This has given rise to the so called *open data movement* which advocates a philosophy that some data should be available to the general public for use in whatever manner it sees fit. Major sources for open data today include research institutions such as the *National Human Genome Research Institute*; which has opened up the results and research data covering their *Human Genome Project*, as well as government bodies which now host data driven portals such as the European Union's *Open Data Portal* and the United States' *Data.gov* initiative. Despite the massive scale and research potential of these data sources and others like them, a major weakness that they share is the fact that they are not linked. This makes them difficult to query and process such that their maximum potential can be extracted using existing tools and techniques.

This thesis describes a holistic environment for data row processing by focusing on a mechanism for interlinking distributed data sources as well a a processing layer to operate on retrieved data.

The thesis is in the English language and contains 70 pages of text, 4 chapters, 40 figures, 3 tables.

Annotatsioon

On täheldatud, et tänapäeval kasvab andmete hulk, mida inimkond salvestada suudab kiiremas tempos kui maailmamajandus. See protsess on käivitanud niinimetatud *avatud andmete liikumise (open data movement)*, mis propageerib ideed et osa neist andmeist peaks olema kõigile avalikult kätte saadavad ja kasutatavad ükskõik mistahes eesmärgil. See on ajendanud nii suuri teadusasutusi nagu *Rahvuslik inimgenoomi uurimise instituut (National Human Genome Research Institute)* kui ka valitsusasutusi nagu näiteks Euroopa Liidu *Avatud andmete portaal (Open Data Portal)* või Ameerika Ühendriikide *Data.gov* avalikustama enda toodetud andmeid veebis. Hoolimata nende ja ka teiste, sarnaste, andmeallikate tohutust andmekogusest and potentsiaalist teadusuuringutele jagavad need kõik ühist puudust - fakti, et need andmeallikad ei ole omavahel ühendatud. See aga muudab nende toodetud andmete pärimise ja töötlemise viisidel, mis nende potentsiaali maksimaalselt ära kasutataks, olemasolevate tööriistade ja tehnikatega keeruliseks.

Käesolev magistritöö kirjeldab andmeridade analüüsimiseks mõeldud keskkonda, mille eesmärgiks on pakkuda robustset mehhanismi ühendamaks erinevaid avatud andmete allikaid ja kirjeldada usaldusväärset, lihtsat ning efektiivset platvormi andmete levitamiseks ja analüüsimiseks. Kirjeldatud platvorm on realiseeritud kasutades P2P arhitektuuri, loomaks omavahel ühendatud sõlmedest võrgu, kus andmete allikas on välja abstrakteeritud kasutades laiendatavat rakenduste raamistikku, mistõttu saavad kõik sõlmed teha päringuid teadmata midagi päringu sisuks olevate andmete allika kohta.

Selles magistritöös uuritakse eespool mainitud süsteemi ahitektuuri, kirjeldades selle peamisi komponente, mis on toodud allpool:

1. Esmalt keskendutakse võrgusõlme olemusele ning sellele, kuidas see ülejäänud võrguga suhtleb.

2. Järgnevalt tuuakse ära mehhanismid, mida sõlmed omavahel informatsiooni vahetamiseks kasutavad. Samuti kirjeldatakse meetodeid, kuidas taotlused ja nende vastused võrgus liiguvad.

3. Edasi kirjeldatakse antud süsteemi potentsiaalseid kasutusalasid andmete analüüsis.
4. Viimaks implementeeritakse, kontseptsiooni tõestamiseks, näidisrakendus mida kasutades tehakse experimente, uurimaks Eesti vee infosüsteemist saadud andmeid ja hindamaks nende ebamäärasuse ulatust.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 70 leheküljel, 4 peatükki, 40 joonist, 3 tabelit.

Table of abbreviations and terms

API	Application programming interface
EEIC	Estonian environmental information center
DB	Database
DBMS	Database management system
DHT	Distributed hash table
HGP	Human genome project
HTML	Hypertext markup language
HTTP	Hypertext transfer protocol
HTTPS	Hypertext transfer protocol secure
IETF	Internet engineering task force
JSON	JavaScript object notation
JSR	Java specification requests
JVM	Java virtual machine
NoSQL	Non SQL
ORM	Object relational mapping
P2P	Peer to peer
REST	Representational state transfer
RFC	Request for comments
SETI	Search for extraterrestrial intelligence
SQL	Structured query language
TCP	Transmission control protocol
UML	Unified modeling language
UUID	Universally unique identifier
YAML	YAML ain't markup language, *Yet Another Markup Language*

Contents

List of Figures

List of Tables

Introduction

Modern computing heavily relies on data. It is one of the humanity's fastest growing resource[1]. The amount of new data being created is much faster than Moore's law meaning that it more than doubles every year[2]. Value is extracted from data through processing to extract meaningful information and metrics. Research institutes as well as businesses over the years have invested heavily in data analytics and such investments have continued to grow along with the data explosion with businesses now investing more in data science and big data than they ever have[3].

The Open definition[4] defines *open* as 'promoting a robust commons in which anyone may participate, and interoperability is maximized'. By this definition, open data is 'data that can be freely used, re-used and redistributed by anyone - subject only, at most, to the requirement to attribute and share-alike'[5]. Open data encourages an unprecedented level of transparency where governments such as the US and EU among others, make public data available for anyone use[6][7]. On the other hand, despite the fact that these initiatives are extremely valuable sources of data, they are not linked and exist as separate collections of datasets. While a lot of research effort has gone into linking this data[8][9][10], it has mainly been focused in specialized domains[11] and thus has not really caught on.

This paper proposes and describes an environment which is aimed at data row analysis. It aims to provide a robust mechanism for linking together open data sources while delivering a reliable, easy and efficient platform for distributing and analyzing data. It utilizes a P2P architecture to build a network of interconnected nodes capable of sending and receiving data across the network which uses an extensible application framework in order to query and process data. The architecture of the system is explored by examining its main components as outlined below:

1. The Node architecture which focuses on what a node is as well as how it interacts with the network. It also focuses on mechanisms in which a node discovers other nodes on the network. This also goes forward in describing the means through which the level of data source access can be abstracted in order to support multiple kinds of data sources as well as the query mechanism. Finally, the application framework is discussed which is the means through which the system can be extended and customized.
2. The communication and data transfer mechanisms used by the application is dis-

cussed which explores the means through which Nodes communicate with each other as well as how requests a propagated between nodes as well as the mechanism through which responses are delivered.

3. The data analytic application of the system are brought into focus by describing a potential use case of the system through an analytical methodology which is used in the experimentation and concluded through an exploration of the practicality of the proposed system.

4. The system is implemented as a proof of concept application and experiments are performed on open data open data particularly the Estonian Water Information System database in an attempt to validate the system. The goal of the experiment is to determine the level of uncertainty[12] in the data as well as determine a margin for error in measurements which were taken while generating the data. The results of the experiments as well as other potential applications and are discussed.

1. System architecture

Modern highly scalable systems today depend on some form of network connectivity to perform a variety of tasks. These means of connectivity include networks such as the internet, wired/wireless telecommunications networks or internal networks in business enterprises. While they all share the general concept of message passing over a network[13], the arrangement of components which participate in the network may impact the overall property of the system as well as the role that they play. For this reason, in constructing a network of this nature a network system architecture[14] must be selected. Based which assigns participants roles based on their function or status. Such architectures include[15]:

1. The Client-Server model

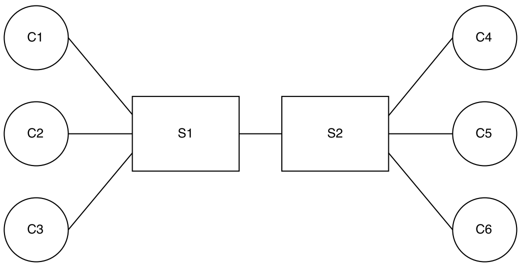


Figure 1. *Client server model*

This network architecture divides network participants into two distinct roles. The *Server*, whose role is to provide/serve functionality to other devices, referred to as *Clients* using the network[16] as the medium of exchange. Functionality often referred to as services rendered by the server vary between applications and implementation, but usually run along the lines of providing information to clients as is the case with web servers serving web pages and traditional RESTful APIs[17]. Servers are also often used to offset workload from clients as they typically have more resources available to them such as additional computing power or the backing of data centers with incredible amounts of data storage available. Client–server systems in place today are often implemented with a request–response model. In this model, clients actively send requests to a server, which in turn performs a task and respond either as a way of acknowledging that a requested task has been completed or to provide information. As depicted in figure 32, the Client-server functions as a centralized model with a one to many relationship between server and client; where one server can have multiple clients connected to it at any point in time. Also depicted in figure 32 is that servers can also connect to other servers. This is a commonly used extension of the client-server architecture known as *Server-to-Server* communication. This allows servers to relay or share information between each other such as database servers exchanging information with application servers.

Advantages of adopting the client-server model include:

(a) Simplified administration
Administration of data as well as management of the server resources is made easy since the bulk of information is stored in a centralized location. It's also much simpler to monitor and control content shared over the network. It is also possible to record and log requests that clients issue to the server.

(b) Scalability
From the perspective of functional and load scalability, scaling up the deployment infrastructure is simply a matter of allocating more resources to the server with the goal of increasing the server's capacity to perform tasks or store information which ultimately translates to the server being able to respond to requests issued by the client faster.

Despite these advantages, the client-server architecture is not a perfect system and does come with its share of problems. Some of the disadvantages of this model are listed as follows:

(a) Single point of failure
A single point of failure is a defined as a system component which, upon failure, renders a given system unavailable or unreliable. The centralized nature of the client-server model means that the server is the backbone of the network. Nearly all data exchanged is routed through the server. This architecture exposes a critical weakness in reliability. By taking the server offline, it's possible to render the entire system unusable.

(b) Network congestion
A server typically responds to the requests from multiple clients often at the same time. It is not unusual for these requests to number in the hundreds of thousands every second and can be much higher during hours of peak traffic. A server can easily be overwhelmed by the number of requests it receives if the request rates are not controlled. Once overwhelmed by traffic, the server may fail to acknowledge new requests effectively rendering the network unreliable.

(c) Cost
As the number of clients increases, the load on the server increases proportionally. After a certain amount of load, the server will not be able to reliably handle all of the traffic leading to network congestion. To mitigate this, the infrastructure must be scaled up to cope with the increase in traffic. However,

acquisition of new infrastructure also increases costs through acquiring new infrastructure but also maintaining it.

2. The Peer to peer model

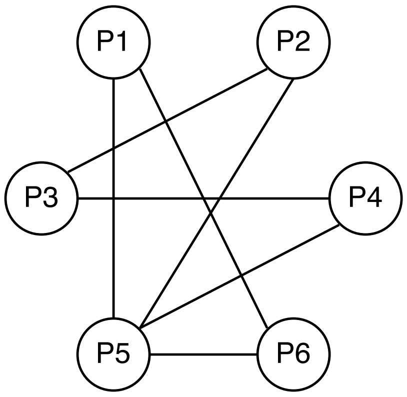


Figure 2. *Peer to peer model*

The Peer-to-peer architecture often referred to as P2P, is a distributed application architecture that aims to partitions tasks or workloads between participants connected to the network known as peers[18]. Peers are considered to be equal participants in the network[19], meaning unlike the traditional Client-server architecture; there is no separation of roles or distinction between computers on the network. Peers typically contribute to the network by making a portion of their resources, such as computing power, storage or simply network bandwidth, available to other peers on the network, without the express need for any kind of centralized arbiter to orchestrate the task[**?**][21]. Peers on the network are considered to be suppliers and consumers of resources on the network, which comes in stark contrast to the traditional client-server model where there is a clear divide between the role of the server as the provider of resources and the client as the consumer of resources. This relationship paradigm creates a system where additional peers contribute not only increase the amount of traffic but increase the scale and stability of the network by contributing resources. This makes it possible to create a massive network of nodes at extremely low costs. Collaborative P2P systems such as SETI@home[22] go beyond simply sharing resources. SETI@home uses its P2P network as a distributed computing platform[23] where peers donate computing power in the search for extraterrestrial life. The SETI Institute has seen a lot of benefit in creating its virtual community of users because empowering them with the ability to collaborate remotely has allowed them to accomplish tasks that no individual peer can do alone.

Advantages of the P2P architecture include[24] but are not limited to:

(a) Highly scalable

Since peers that join the network contribute their resources, the effective network infrastructure is scales as well with the lower cost of deployment. There is also no need for a single administrator maintaining the resources. Being distributed, peers are often responsible for self-administering their infrastructure.

(b) Reliable

Since peer to peer networks have no concept of a central server, there is no single point of failure on the network. Taking down individual peers will not disrupt the entire network.

(c) Distributed work load

Since the load on being transmitted over the network isn't concentrated at single points in the network, it is better distributed among peers. This better utilizes resources on the network contributing to its scalability.

As with all things, in order to maintain a holistic view of the system, the disadvantages must also be considered.

(a) Difficult administration

It's often tough to manage the network since often peers may not be on the same continent yet remain critical nodes contributing to the stability of the network. Unlike the Client-server architecture, a single person may not be able to determine the entire reach of the network to put things like upgrades into effect.

(b) Security

It's often difficult to know what is transmitted over the network beforehand. Peers have little choice other than to implicitly trust other peers to exchange data successfully. While some measures exist to mitigate this problem, like checksum verification and virus scanning. Not having a centrally managed server that can be audited makes this endeavor challenging and expensive.

(c) Data backup and redundancy

Unlike having a central server there is no simple way to guarantee that copies of information transmitted over the network will be made. Therefore, it's imperative that peers self-enforce their data backup and recovery strategy.

With these network architectures in mind, the Client-server architecture is not ideal for building out networks at scale on a budget. It requires a massive infrastructure deployment to support scores of clients; connected data sources should also be attached to the server in ideal circumstances. The P2P architecture, on the other hand, sees peers as equal participants meaning that they are all responsible for their data sources as well as resources and infrastructure. For this reason, P2P networks scale well in low-cost situations. This makes the P2P architecture ideal in designing this system.

1.1. Node

This system defines a node as *a single operating instance of the software running on a device*. This is analogous to a peer in the P2P architecture. From a purely functional perspective, it functions as both as a client (requesting and receiving information) and a server (honoring requests for information). Nodes may be added to the network by connecting establishing connections with other nodes this allows them to issue and receive requests as well as forward requests to other nodes that they know about. Nodes may be connected to data sources from which the may query for information, or serve to other nodes that issue requests. While the presence of a data source is ideal, it is not a strict requirement. Nodes that do not have data sources still retain all of the functionality available to other nodes and may issue, receive and forward requests. Such nodes are referred to as *forwarding nodes*.

Nodes are not merely limited to being simple interconnections but act as modular application containers. This is made possible by the comprehensive set of APIs which they expose to allow interconnection with the system. External applications can implement these APIs to augment and extend the system. Added functionality can then be dynamically invoked at runtime. Since these extensions or plugins run within the context of the application, they can be referred to as *plugins* or *Applications*. This raises the reach of the system to becoming an extensible and customizable platform that can be configured to cater to the users needs. Applications can be invoked dynamically at run-time when appropriate activation conditions are met to perform tasks. The exact nature through which this is implemented is discussed in subsequent sections of this paper. With this in mind, the system can be divided up into three parts as illustrated in figure 3,

1. Communication, dealing with the interconnection of nodes, and how they transfer information between each other.
2. Data source management, which provides a set of APIs that raise the abstraction level in dealing with connecting to heterogeneous data sources.
3. and the Application framework, the method in which external applications can be

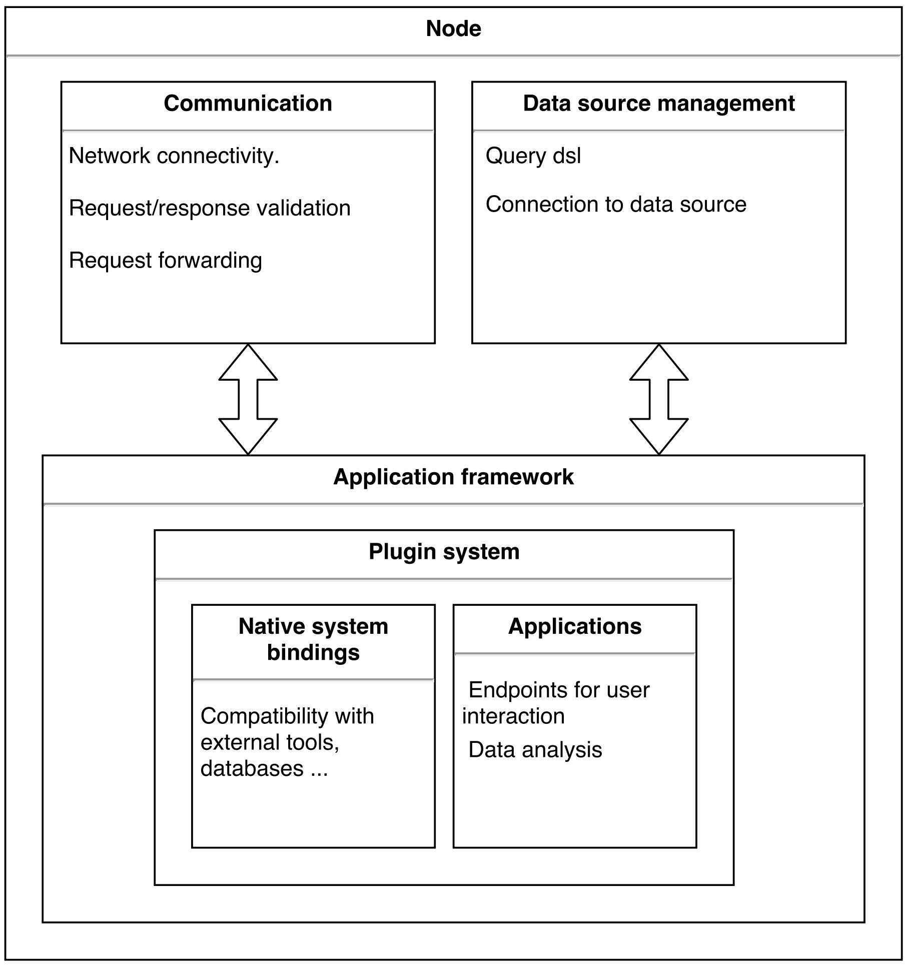


Figure 3. *Node architecture*

run on the node. This layer utilizes the other two in an effort to obtain information from other nodes and process the resulting data.

An important aspect of this system is the model and relationship between nodes. When such relationships are sketched, they appear to mirror relationships between edges (lines) and vertices (points or nodes). As a result, nodes and their interconnections can be modeled using graph theory[25]. A graph is formally defined as,

$$G = (V, E) \tag{1}$$

Where V as a set whose elements are called vertices, nodes, or points and E is a set E of edges, arcs or lines, which are 2-element subsets of V. Graphs can be broadly divided into two categories[26]:

1. A Directed graph, where the edges which connect vertices have direction associated with them (also referred to as a digraph)[27],
2. and an Undirected graph, where edges have no direction associated with them (also referred to as a simple graph)[28].

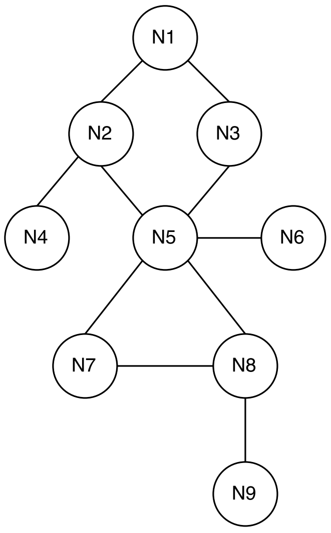


Figure 4. *Graph consisting of 9 interconnected nodes*

As nodes send and receive requests (sometimes simultaneously), the nature of their communications can be said to be in full-duplex mode. Therefore, a graph illustrating these relationships must accurately reflect this, making appropriate to use undirected simple graphs as is the case in figure 4.

Taking this a step further, graph theory helps to define critical properties which help in characterizing the network and relationships between nodes. For example, the maximum number of edges E_{max} can be can be calculated using,

$$E_{max} = \frac{n\,(n-1)}{2} \tag{2}$$

where n represents the number of nodes. Applying this to the graph illustrated in figure 4 gives a maximum number of edges of 36. The density[29] (closeness to the maximum number of edges) can be calculated using,

$$D = \frac{2\,|E|}{|V|\,(|V|-1)} \tag{3}$$

Where E refers to the number of edges and V are the number of vertices (nodes). Substituting the relevant values gives from figure 4, gives a density of 0.2778; indicating that it's not very dense for its size.

The *diameter* of the graph is the maximum distance between any two nodes and represents a worst-case path for relaying information between any two nodes under static conditions[30]. The *degree* of a node is the number of connections to adjacent peers on the graph. For example, in figure 4, node $N6$ has a degree of 5, while leaf-node $N4$ has a degree of 1.

1.1.1. Traversal

Traversing a network/graph is the process of visiting each node on the graph[31]. Fully traversing a network involves visiting/processing each node on the s each node at least once. Some traversal methods add restrictions such as, the number of times that nodes can be revisited and the order that they are initially visited. This opens us up to several means of traversal of the network[32]. For example,

1. Depth-First Traversal
   This algorithm traverses a tree by following the left most branch until it encounters a leaf node (a node with no child branches). It then proceeds to climb back up the branch and visit the closest encountered sub-tree[33]. This is illustrated in figure 5. There are several different types of depth first traversals distinguished by the order in which the nodes are visited:

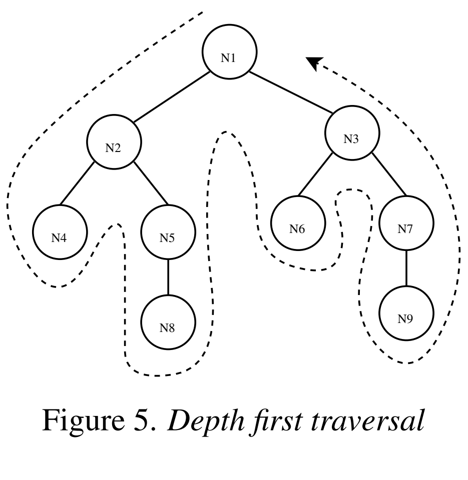


Figure 5. *Depth first traversal*

- The *pre-order* traversal visits nodes before their sub-tree. Applying this algorithm to the graph depicted in figure 5, the nodes will be visited in the order, $N1, N2, N4, N5, N8, N3, N6, N7, N9$.

- The *in-order* traversal visits nodes only after visiting their left sub-trees. By applying this algorithm to the graph in figure 5, the nodes will be visited in the order,
  $N4, N2, N5, N8, N1, N6, N3, N7, N9$.
- The *post-order* traversal visits nodes only after visiting their left and right sub-trees. Once again applying this to figure 2 yields nodes visited in the following order,
  $N4, N8, N5, N2, N6, N9, N7, N3, N1$.

2. Breadth-First Traversal
   This algorithm traverses the tree by starting from the root and visiting its children from left to right and their children, and so on[34].

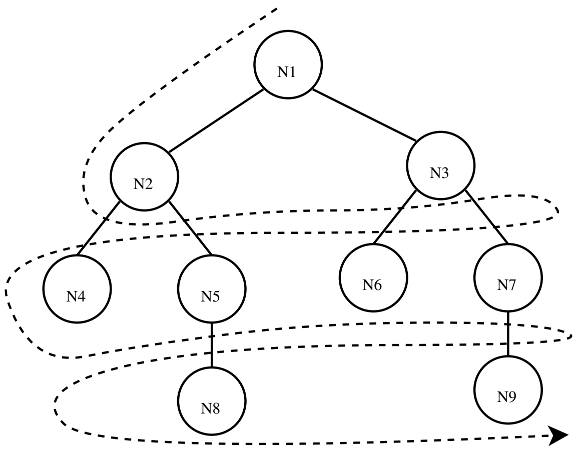


Figure 6. *Breadth first traversal*

Given the graph in figure 6, the nodes will be visited in the following order, $N1, N2, N3 ... N9$.

1.1.2. Networking

Nodes are networked when two or more nodes are interconnected. From a graph theory perspective, this is just adding a vertex onto the graph and connecting it to other vertices using at least one edge. Implementing such a system using a P2P architecture imposes certain requirements on the connection protocol:

- Stateful - The connection must be aware of nodes that have initiated a connection[35]. This allows for requests to be forwarded to connected nodes. In other words, it must not be stateless.

- Bi-directional - Connections made must be capable of both sending and receiving data under normal conditions.
- TCP Support - It must support at TCP connections.

Considering the previous requirements, plain old HTTP/HTTPS connections can't support these connections due to their stateless nature[36]. Additionally, Servers are only able to pass on information to clients in response to requests issued. This results in many resources intensive paradigms such as polling and long polling[37] strategies which ask the server if there are updates at fixed intervals indefinitely, and HTTP/keep-alive connections[38].

A better strategy would be to open network socket connections and send the data directly. Network sockets, often referred to simply as sockets are endpoints for a connection on a network which are bound to an IP address and port number such that the TCP layer can identify the implementing application[39][40]. This maintains a two-way stateful communication environment between a client/server or nodes on a P2P network. While native socket implementations will work perfectly and are incredibly powerful, they are not supported by modern web browsers. For this reason, *the HTML 5 WebSocket protocol* described in the *RFC-6455* specification[42] has been selected for this system. HTML5 WebSockets commonly referred to as simply WebSockets provide bi-directional and full-duplex communication using a single socket connection using HTTP[41]. Along with browser support, HTML5 WebSockets account for common network hazards that make native socket implementation difficult such traversing firewalls and proxies. WebSocket-based applications also place less burden on servers by allowing them to support more concurrent connections.

The WebSocket protocol achieves this by working with existing web infrastructure[43]. This means that his design principle, the protocol specification defines that the WebSocket connection starts its life as an HTTP connection, guaranteeing full backward compatibility with the pre-WebSocket world. The WebSocket handshake protocol is used to upgrade a connection from a simple HTTP connection to a WebSocket connection as illustrated in figure 7 .

1.1.3. Discovery

P2P based systems typically consist of a vast number of distributed, autonomous peers which interact with each other. As a result, they must be able to locate other peers .

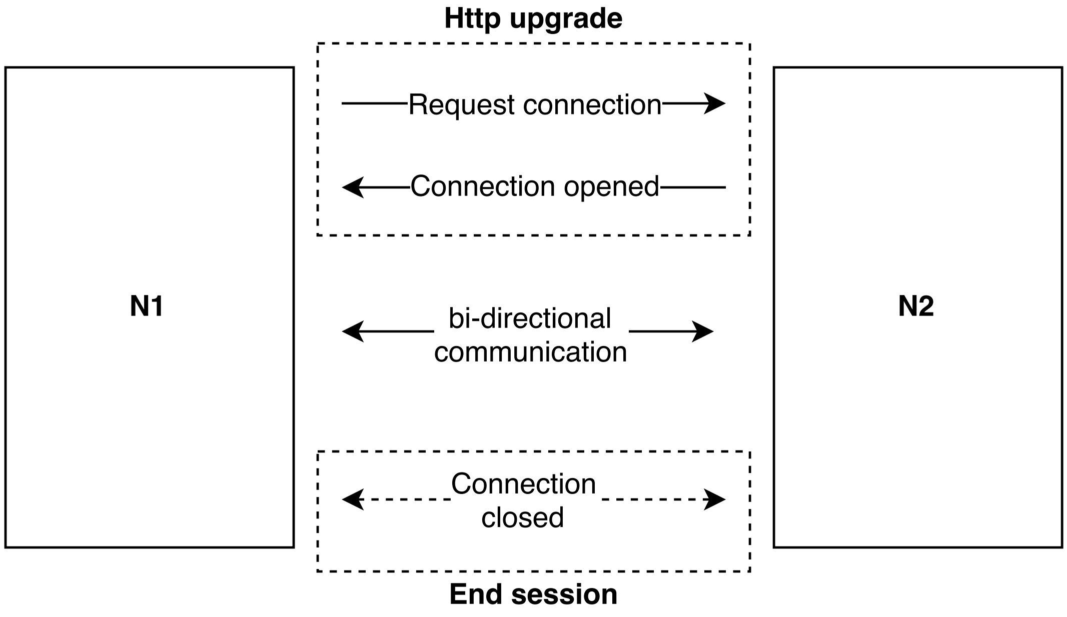


Figure 7. *RFC-6455 connection protocol*

Providing a means of joining/participating in the network which is often more challenging than one would initially assume. With the client-server approach to distributed systems, clients know exactly how to locate servers usually through hard-coded URL endpoints. Since P2P networks often do not use central servers, they require advanced peer discovery methods which allow peers to find, identify and make connections with other peers. These detection mechanisms can be broadly classified into *Centralized* which implement some form of DHT or other types of peer location/metadata caching and *Decentralized* models. These models along with some of their known implementations are discussed below[44].

- Flooding Protocol often referred to as query flooding has peers flood an overlay network with queries aimed at obtaining information about the network[45]. Since peers do not maintain any central directory they resort to discovering other peers by passing along queries through other peers. The robustness and extensive reach of discovery characterize flooding protocols[46]. Flooding protocols tend to work best in networks with a small number of peers thus do not scale well. It also does not guarantee that all peers on the network will be discovered[47].
- Chord is an implementation of a DHT that uses an overlay ring topology. It assigns specific keys to nodes which in turn store values for all keys which they are responsible. The Chord also dictates how nodes will use keys that they have been given in locating other nodes on the network.
- Pastry is an implementation of a DHT routing protocol that is prefix-based and self-organizing. Each node in the Pastry network is issued a unique identifier referred

to as the node id, from a 128-bit circular index space. The Pastry node works by routing messages to the node with the numerically closest id that contained in the message. It repeats this process until the message arrives at its destination.

The flooding protocol is the only alternative listed that does not require an overlay network to communicate with other nodes. It also the only option that does not need a DHT to function. The disadvantage with the flooding protocol is that it is that it can quickly increase the density of the graph. While these alternatives are ideal for building self-organizing networks, it's also possible to design the network such that it requires explicit connections between peers in order to organize the network. In this mode however, peers should still be able to act as forwarding nodes for requests.

1.2. Data sources

Data used in research is often encapsulated in multiple formats. These are further encapsulated in multiple databases which most common of which include but are not limited to the following.

- Relational databases
  These are databases where data is organized into tables containing rows and columns. Commonly referred to as SQL databases[49], tables contained, often cross-reference each other through unique keys applied to each table. These tables are managed using DBMS systems to modify data contained in rows[50].
- Non-relational databases
  Often referred to as NoSQL databases, these include database structures such as Document or key-value stores as well as Graph databases[51].
- Files
  These are basic implementations of structured data contained in a unit on a storage medium. These may either be directly manipulated by the user or through applications running on the system. Popular formats used to store data include, CSV and spreadsheets[52].

Since nodes act as mediators between requests for data and the actual data sources, it's important for the robustness of the system to support as many formats as possible. This is a challenging proposition even within just the scope of relational database implementations. This is because different vendors tend to implement the SQL specification differently[53]. This concept of divergence from a common universal schema is further exacerbated when comparing SQL and the various NoSQL databases[54] as well as the arbitrary nature of files that may exist on the users file system. Any system that aims to support these many Provide support for these forms of data sources means that it in a practical way, it cannot be built into the system. This is because the various schemas and standards (or lack thereof in the case of arbitrary data files) implemented may change over time. Integrating them into the system means that it must evolve at the same pace in order maintain support. This problem can be solved by abstracting support for data sources, out of the system and integrating it into the application layer.

1.3. Query abstraction

Developing a data format agnostic query environment is a challenging endeavor[55]. It requires adequately separating the specification and realization layers involved in the task being accomplished while managing all of the hidden layers. Despite the difficulty, abstraction of data source query has been implemented to some degree. For example, ORM tools attempt to raise the level of abstraction for SQL databases. Popular ORM tools such as Hibernate[56] and LINQ to SQL[57] do this quite well. Taking advantage of such tools as a way of abstracting queries between multiple databases, ROOMA (Jaan Aigro, Vladimir Viies. 2014)[58], act as a bridge service between multiple DBMS. The result was a system that can query multiple databases and perform some computation on the resulting data[59] which is processed by the system and fed back into the database.

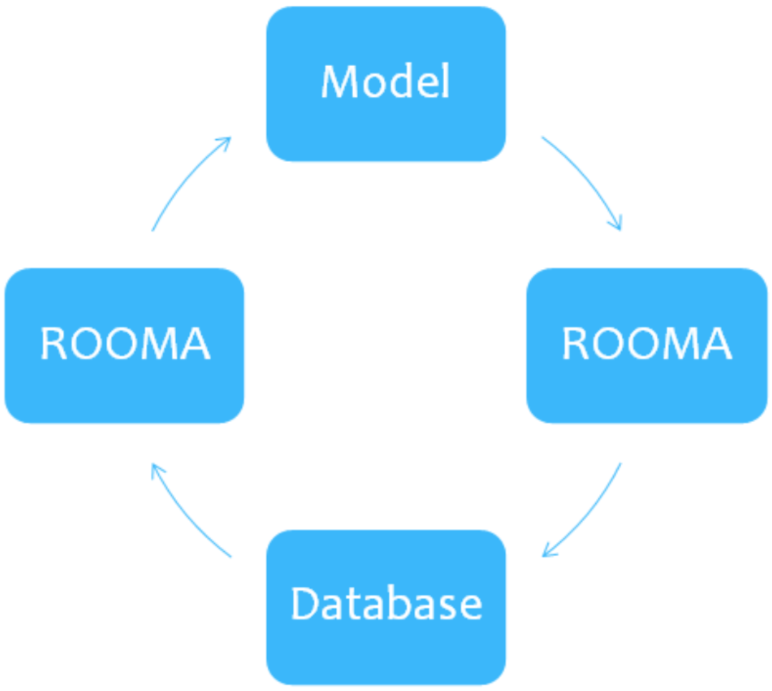


Figure 8. *ROOMA work flow for running model[58]*

While this was an effective way of solving the given problem, it comes with its fair share of challenges when studied in detail particularly with regards to scalability,

1. Since support for compatible DBMS are hard-coded into the application, the source code must be modified to support new systems.
2. ORM tools are bound to the platform on which they are written and tend to behave differently depending on the programming language.
3. The solution does not cover other forms of data sources. Non-DBMS solutions are not supported.

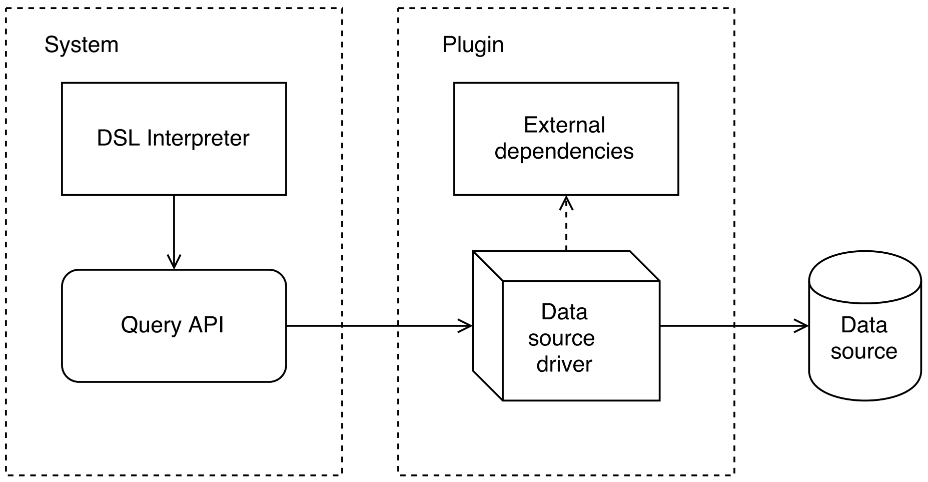


Figure 9. *Query system structure*

A more practical approach to solving this problem would be to raise the level of abstraction and introduce a domain specific language for querying data sources which invoke a set of APIs which abstract common query functionality create bindings here referred to as the *data source driver* for their own data sources. The benefit of having the package decoupled from the system is because it can be maintained separately. This domain-specific language can be SQL-like or adopt the familiarity of more general purpose languages for enhanced flexibility in data manipulation. The data source driver is expected to communicate with the users DBMS system or file system either directly or indirectly through other subsystems that may be available to the user.

1.4. Application framework

Modern software has seen an increase in complexity over the years. It's said to be some of the most complex artifacts ever created by humans[60]. The cause of this level of complexity may be attributed to the level of expectation that we've grown accustomed to with the stability and reliability that some of these systems have demonstrated such as, modern fly-by-wire systems. As a result of these advancements, it's tough to manage the functional complexities that these systems carry.

The success and increasing popularity of open-source software over the years has made library reuse a rather compelling part of developing sophisticated software. Many new programs today use combinations of open source libraries, in order to solve new problems at no monetary cost. Splitting up an application into smaller modules also improves the quality of the software from both the programmers and user's perspective. Big monolithic pieces of code become increasingly interconnected (where lines of code in a source file access lines of code in other source files) and obscure as described by the anti-pattern *Spaghetti code* or *blob code*[61]. Having a lot of ultimately makes the code extremely complicated and difficult to maintain by those unfamiliar with the overall structure of the system. To combat this, it is important to break up functionality into modules. Where .By design, the system must be modular. This allows for larger extremely more comprehensive solutions to be implemented and separately maintained from the application server[62].

2. Data Transfer

2.1. Architecture

An important part of the system is rooted in its ability to easily and reliably pass data between nodes. This allows them to exchange data which can be analyzed to extract meaningful value later. It achieves this through an elaborate data transfer structure that is integrated with the application framework (Appendix 1, figure 15 illustrates the entire process).

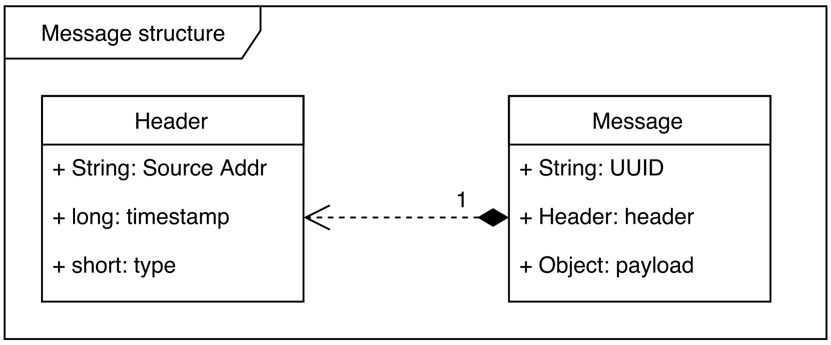


Figure 10. *Structure of a message*

A message consists of metadata directly related to the context of its use as well as a payload as illustrated in figure 10, this includes but is not limited to:

- A unique identifier is used by nodes to determine whether a request has not has been previously handled. A list of Ids that have been handled are cached, and subsequent requests or otherwise that contain Ids in the cache are given an appropriate response or are simply ignored. For this purpose a standard practice involves the use of a UUID as described in the RFC 4122 specification[63] and standardized by the Open Software Foundation as part of the Distributed Computing Environment effort. A UUID is simply an arbitrary 128-bit value. It is often rendered for human readability as a string of hexadecimal characters with hyphens inserted partitioning the values.

For example,

```
dc0e4c39-986f-48ac-8668-330dca697ca8
```

Figure 11. *A Sample version 4 UUID*

While it's impossible to guarantee completely that the values will be unique, the use of a UUID ensures that the probability of a hash collision occurring (where the same value is generated more than once) is extremely improbable[64].

- The Message header contains information about the source node, as well as a value which indicates the type of message that is being passed. These values could correspond to information such as *configuration, request, response... etc*. The receiver of the message is expected to check the message header to determine how the payload should be processed.
- The Payload is the core information that is to be transferred.

Transmission of objects/data over the network often requires that data be serialized and deserialized[65]. Serialization is the decomposition of complex data structures such as objects, database containers, etc, which are actively used in the ongoing operation of the application into a sequence of their primitive data components representing the objects state as well as the object model describing the object[66]. This operation is performed with the intention that this object can be recreated at a later date. Sequences of data obtained as a result of serialization can then be directly written to a file for storage or transmitted over the network. The process of re-creating complex data structures from serialized data is known as deserialization. Deserialization can be performed on either on the host machine/software that serialized the data or other machines/software to which the serialized data was transferred.

Standard serialization encoding options today typically opt for text based formats. This offers many advantages because they can be easily read by humans, and are programming language and system agnostic; meaning that they can easily be transferred and worked on using nearly and language or system. Standard formats in use today include:

- XML[68], is a markup language that is designed to be both human and machines. The design goals of XML was to emphasize simplicity, and readability when transferring data across the Internet. This has made it a very common format that encourages interoperability with other tools but does not have very good performance due to the excessive amount of object data stored in tags.

```
<Message>
  <UUID>dc0e4c39-986f-48ac-8668-330dca697ca8</UUID>
  <Header>
      <SourceAddr>127.0.0.123</SourceAddr>
    <timestamp>123456789</timestamp>
    <type>2</type>
  </Header>
  <Payload> ... </Payload>
</Message>
```

Figure 12. *Snippet of serialized message rendered in XML format*

- YAML[69] aims to be an extremely concise human readable language that is more data, than object oriented. It favors indented delimiting as a way of distinguishing between different sets of data and is limited to hierarchical data with each node having a single parent.

```
Message:
  UUID: "dc0e4c39-986f-48ac-8668-330dca697ca8"
  Header:
    SourceAddr: "127.0.0.123"
    timestamp: "123456789"
    type: "2"
  Payload: " ... "
```

Figure 13. *Snippet of serialized message rendered in YAML format*

- JSON[70] is a subset of JavaScript which was introduced as a lightweight alternative to XML for transmitting content in the browser. It features a more concise and human-readable structure with significant community adoption and support. It is a developer favorite when building web services such as RESTful APIs.

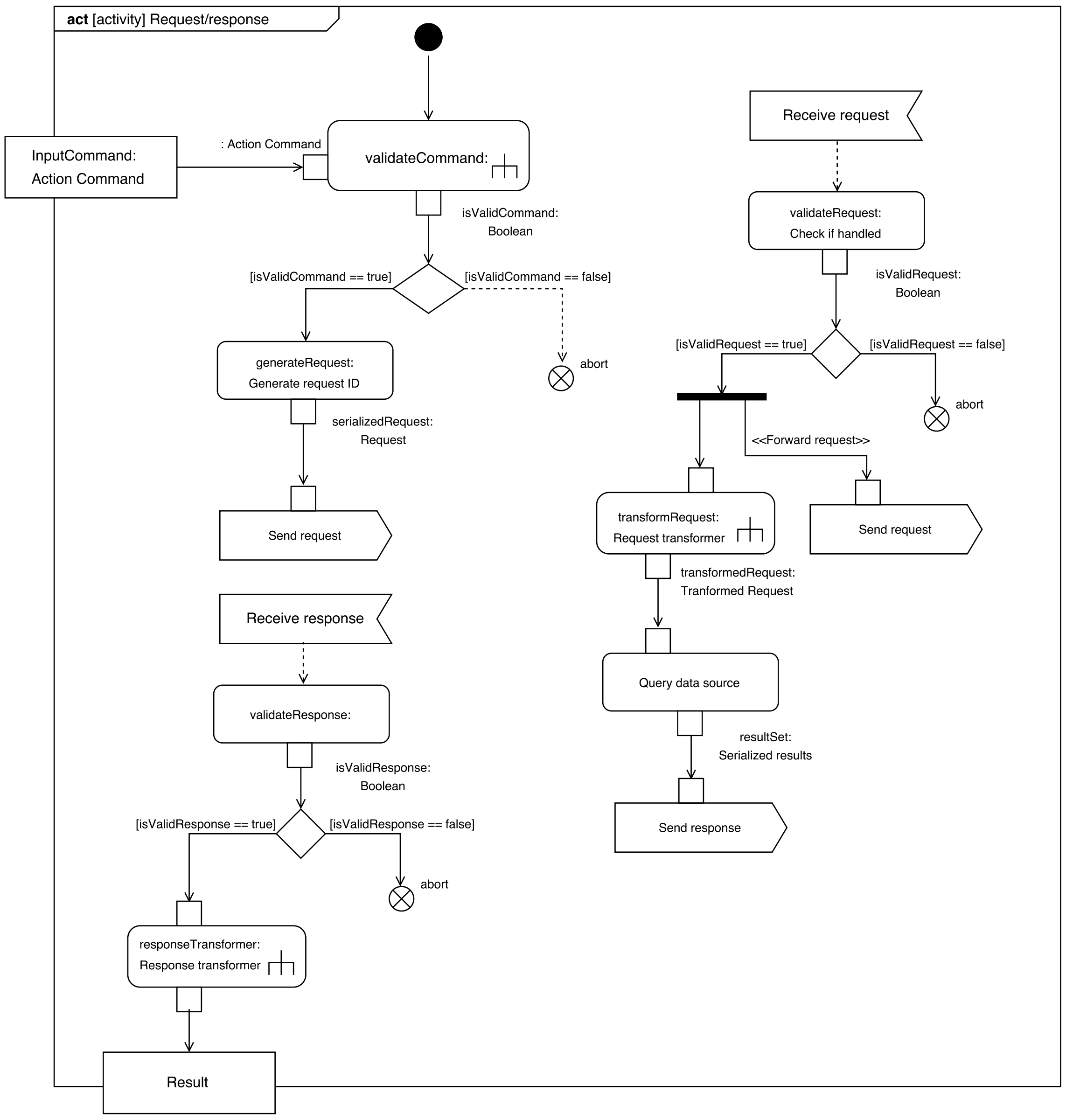
While it is possible to use any serialization and deserialization strategy in the system implementation, the JSON format was selected to due to its high adoption rates and browser support. It's concise nature, and strong community has manifested itself through an active library ecosystem with mature libraries for manipulating large structured JSON across many languages and frameworks.

```
{
  "Message": {
    "UUID": "dc0e4c39-986f-48ac-8668-330dca697ca8",
    "Header": {
      "SourceAddr": "127.0.0.123",
      "timestamp": "123456789",
      "type": "2"
    },
    "Payload": " ... "
  }
}
```

Figure 14. *Snippet of serialized message rendered in JSON format*

2.2. Request

Requests are sent between nodes as a way or asking for information. Requests are encapsulated as messages using a corresponding type specifier and as a result, are serialized for transmission over the network. Requests are issued as a result of user input specifying a query to be performed and are broadcast as a message containing a query as its payload.



act [activity] Request/response
InputCommand: Action Command
: Action Command
validateCommand:
isValidCommand: Boolean
[isValidCommand == true]
[isValidCommand == false]
generateRequest: Generate request ID
abort
serializedRequest: Request
Send request
Receive response
validateResponse:
isValidResponse: Boolean
[isValidResponse == true]
[isValidResponse == false]
abort
responseTransformer: Response transformer
Result
Receive request
validateRequest: Check if handled
isValidRequest: Boolean
[isValidRequest == true]
[isValidRequest == false]
abort
<<Forward request>>
transformRequest: Request transformer
Send request
transformedRequest: Tranformed Request
Query data source
resultSet: Serialized results
Send response

Figure 15. *Activity diagram detailing the request/response process*

The process for issuing and handing requests is described in figure 15. It begins with the user inputing a command into the system to start an action, this is going to be a query of some kind. After validation the request is broadcast on the network. Receiving nodes take the request and put it through the transformer as it relays the query to whatever data source and data access solution is in place. Once the query is completed, the response is issued which is similarly put through the response transformer before the data is processed.

2.3. Request propagation

A core function of the system is propagating requests for data across the network using a node traversal technique described in an earlier chapter. This allows nodes to request and exchange information. There are several methods in which requests may be propagated[18]:

1. Centralized directory model

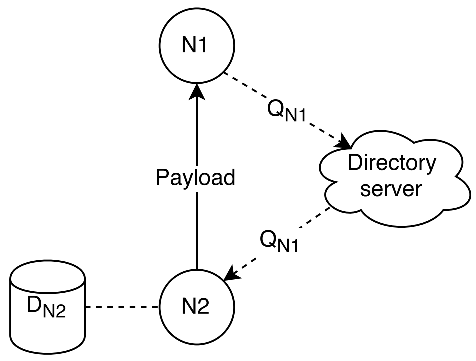


Figure 16. *Centralized directory model*

   In this model, a dedicated node maintains a record of all nodes available on the network as well as information about the kind of data that nodes are serving. A request for information is first issued to the directory server. The server matches the content of the request and forwards it to suitable nodes. Nodes that have been sent the request then respond by opening a direct connection the request originator and sending the response. Since all nodes on the network need to rely on the directory server, it may become the network bottleneck.

2. Flood requests model

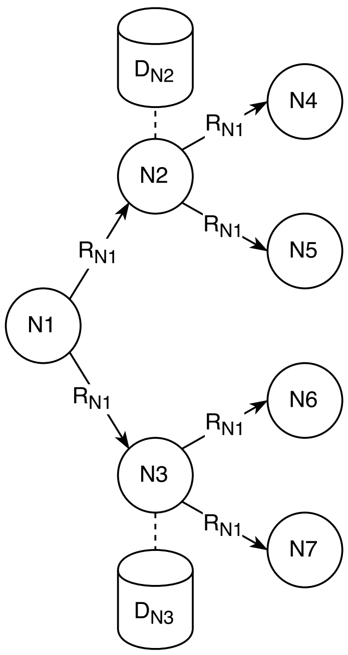


Figure 17. *Flood requests model*

This is similar to the flooding discovery protocol where peers directly broadcast requests to their neighbors on the network to locate data. Peers may reply or relay the request to their own directly connected peers. This may be unlimited in its ability to reach every node on the network or limited by a given number of hops or flooding steps

2.4. Response delivery

When a node receives a request forwarded from other nodes on the network, it acts by processing the request and querying its data source as discussed in other chapters. However, there are certain complexities associated with delivering its response which must be addressed. Such complications include the manner in which it takes place concerning the architecture of the network.

1. Reverse response propagation

   In this model, responses are propagated along the same path from which they are requested. This means that the chain through which requests come through is respected and nodes strictly communicate only with nodes where they share direct connections.

   Figure 18 illustrates this mechanism by considering a network of 6 nodes with 5 of them connected to data sources. We assume that the originator of the request is

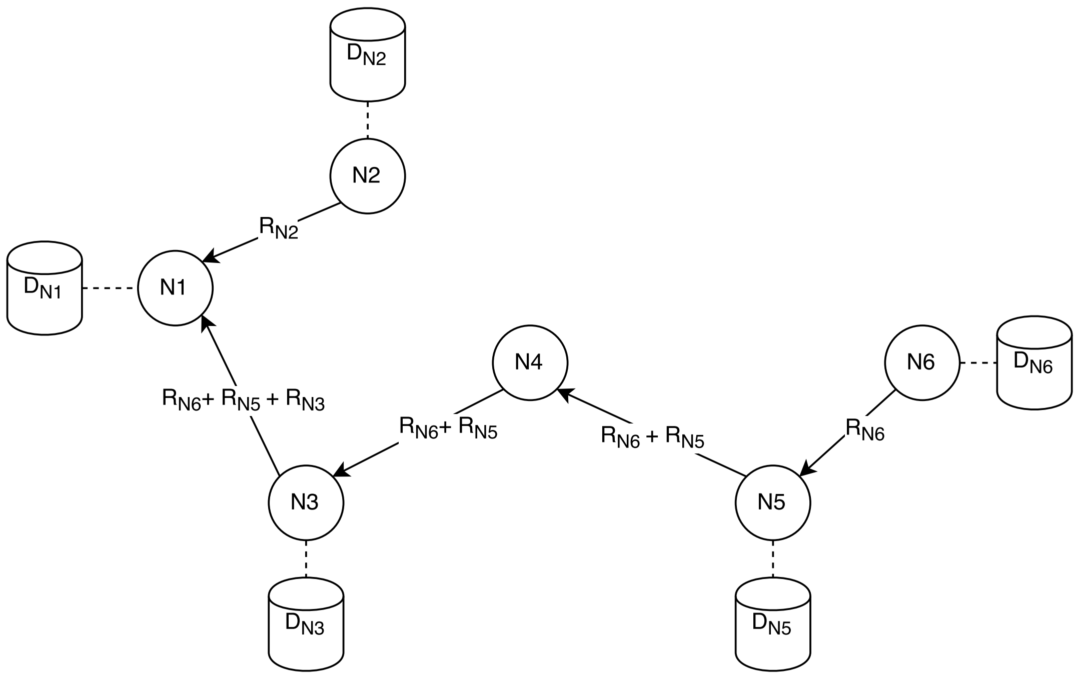


Figure 18. *Reverse response propagation. 6 nodes propagate responses their responses to requests issued*

$N1$ using the flood requests model. Through request propagation and the method in which the network is traversed, $N6$ will eventually receive the request and query its data source D_N6. Once this is completed, it will issue it's response back along the route in which it was received.

It has the advantage of maintaining the philosophy of only ever communicating with a direct neighbor thus maintaining some degree of anonymity between peers. While this is a reasonable method of addressing the problem, it does have some severe limitations,

(a) It requires a lot of bandwidths because each time a node is visited, the response data is propagated back. This is seen in figure 18 where the response data R being returned by $N3$ is the given as $R_{N6} + R_{N5} + R_{N3}$. This means that on a large scale deployment, this becomes impractical with nodes relaying data from hundreds/thousands of other connected nodes.

(b) It also requires that nodes wait for other nodes further down the chain to respond before transferring their data. This time accumulates meaning that the amount of consumed waiting for responses is a factor of the slowest node on the network.

2. Direct connection response

The direct connection response method attempts to solve the scalability issue of the reverse response propagation method by enforcing that nodes make direct connections to the request originator in order to deliver responses.

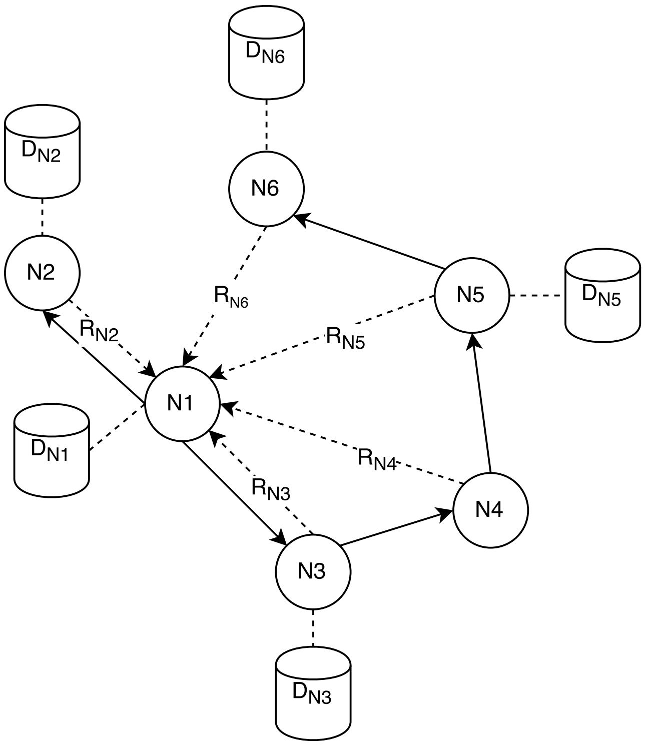


Figure 19. *Direct connection response mechanism*

Figure 19 illustrates this concept, using the same network structure as that of figure 18; meaning that $N1$ is the originator of the request which is propagated using the flood request pattern. Rather than propagate responses back the path that they were issued, responses are issued directly back to $N1$.

While this addresses the major scalability concerns raised, it introduces other problems.

(a) There is no way to know how many responses to expect or how long the request originator should wait for responses.

(b) An excessive number of nodes sending responses may overwhelm the request originator.

(c) In breaks the model of nodes only communicating with neighboring nodes. This could potentially be a security vulnerability.

2.5. Request and response transformation

In order to pass requests to and from plugins on the application layer. The system uses a concept of request and response transformation. The goal of which is to make data usable for plugins before and after deserialization.

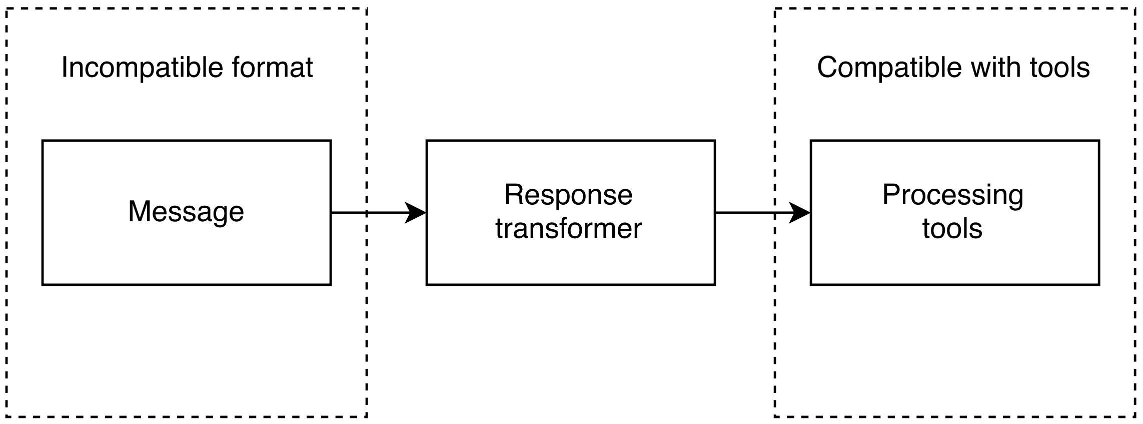


Figure 20. *Response transformation process*

When data is deserialized, it is deserialized to the Message format, this means that any tool that wants to process the received data must be compatible with the message format. Unfortunately this isn't always the case. For this reason, the system exposes an API which is invoked whenever results are to be passed to plugins. This is the response transformer as shown in figure 20. Plugins are expected to implement these transformers in order to work effectively with the system.

3. Data analytic applications

The system's modular architecture means that it is very extensible and can be configured to run applications that target the framework. These applications do not necessarily need to be written using the same programming of the system. The goal of plugins is to extend the functionality of the system and add new features. They accomplish this by extending the functionality of the request/response transformation mechanism exposes APIs to a plethora of analytic applications. A possible analytic application is explored and demonstrated in subsequent sections.

3.1. Benford's Law

The digits in numerical data produced by a large number of very different natural and social processes take the form of a logarithmic distribution described by Benford's law. Given the number and variety of processes that produce Benford-distributed data, it is often assumed that the first (i.e. the leftmost, regardless of the position of the decimal point), second and later significant digits in many kinds of real numerical data adhere to Benford's law. The additional assumption that fabricated or falsified data are detectable through the deviation of their digits from the Benford distribution has been tested recently in several contexts. For example, some studies have reported success in identifying fraudulent information with a check of digital frequencies in tax or other financial data against the Benford distribution. Similar results have been reported for fabricated survey interviews

Benford's often referred to as the *first digit law*, is an observation that in many collections of numbers, mathematical, real-world statistics or otherwise, the digits are not uniformly distributed.

Formally, Benford's law states that the probability of the first signifigant digit D_1,

where,

$$D_1 \in \{1, ..., 9\} \tag{4}$$

occurs with probability,

$$P(D_1 = d_1) = \log_{10}\left(1 + \frac{1}{d_1}\right) \tag{5}$$

The probability of the second digit D_2 where,

$$D_1 \in \{0, ..., 9\} \tag{6}$$

occurs with probability,

$$P(D_2 = d_2) = \sum_{d_1=1}^{9} \log_{10}\left(1 + \frac{1}{d_1 d_2}\right) \tag{7}$$

The application of this formula reveals the probability that the first digit equalling one rests at approximately 30%.

Table 1. Benford's law expected frequencies

Leading digit	**First place**	**Second place**
0		12.0%
1	30.1%	11.4%
2	17.6%	19.9%
3	12.5%	10.4%
4	9.7%	10.0%
5	7.9%	9.7%
6	6.7%	9.3%
7	5.8%	9.0%
8	5.1%	8.7%
9	4.6%	8.5%

The quantity $P(D_1)$ is proportional to the space between D_1 and D_1 + 1 on a logarithmic scale. Therefore the Benford distribution is the expected if the numbers themselves are not uniformly distributed however the mantissae of the logarithms of the numbers are uniformly and randomly distributed. The expected frequency of occurence of numbers that exibit a Benford distribution are given in table 1.

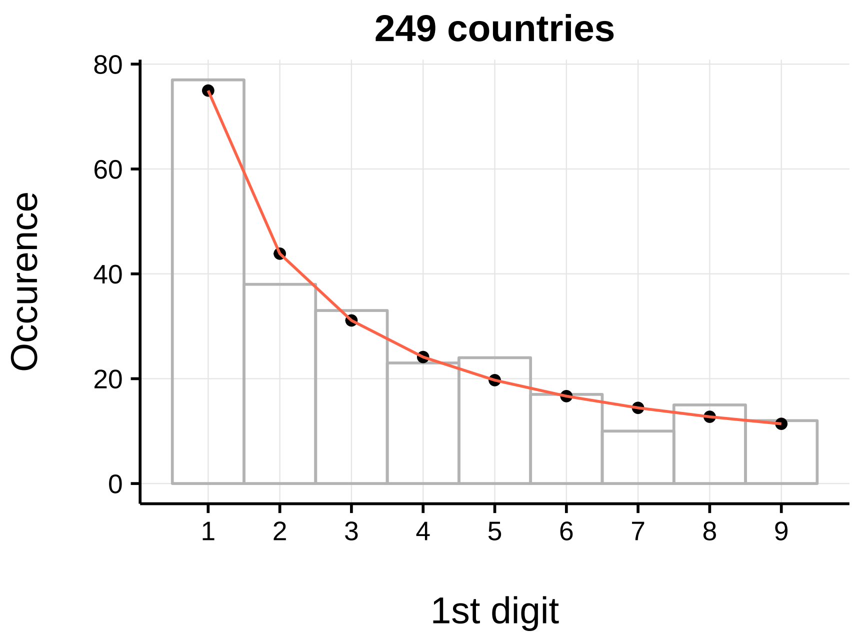


Figure 21. *First digit distribution of population of 249 countries plotted against its Benford estimate*

Benford's law can be observed figure 21 it's clear that multiple data sets conform to Benford's law, not all set of numbers do.

Benford's law earned a title as a one of the gems of statistical folklore. It is the observation that in many collections of numbers, be they mathematical tables, real-life data, or combinations thereof, the leading significant digits are not uniformly distributed, as one might expect. Instead, the digits tend to be logarithmically distributed and are heavily skewed toward the smaller digits.

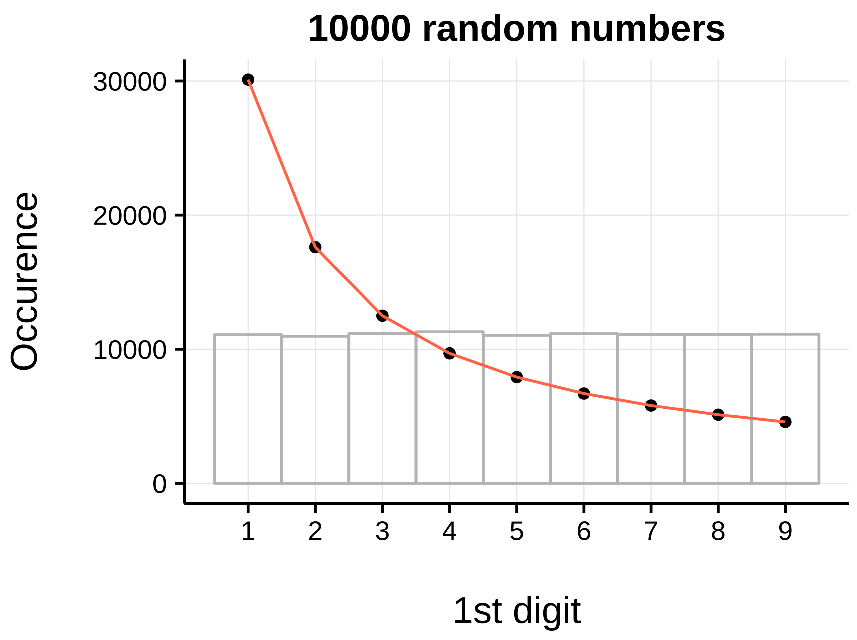


Figure 22. *First digit distribution of a random sequence*

For example, the histogram illustrated in figure 22 was generated from a data set consisting of 10000 values generated using a random number generator. The overlaid dots indicate the expected values for the histogram to conform to a Benford distribution.

3.1.1. Scale invariance theorem

Given data-sets where that are usually bound by reasonable ranges, such as a list of lengths, the distribution of first digits that these numbers typically have in the list may be generally similar regardless of their unit of expression (For example, in meters, yards, feet, or inches). For example, the height of adult humans almost always starts with a 1 or 2 when measured in meters, and almost always starts with 4, 5, 6, or 7 when measured in feet.

But consider a list of lengths that is spread evenly over many orders of magnitude. For example, a list of 1000 measures mentioned in scientific papers will include the measurements of molecules, bacteria, plants, and galaxies. If one writes all those measures in meters or writes them all in feet, it is reasonable to expect that the distribution of first digits should be the same on the two lists.

In these situations, where the distribution of first digits of a data set is scale invariant (or independent of the units that the data are expressed in), the distribution of first digits is always given by Benford's Law. To be sure of close agreement with Benford's law, the data has to be approximately invariant when scaled up by any factor up to 10. A lognormally distributed data set with wide dispersion has this approximate property, as do some of the examples mentioned above.

For example, the first (non-zero) digit on this list of lengths should have the same distribution whether the unit of measurement is feet or yards. But there are three feet in a yard, so the probability that the first digit of a length in yards is 1 must be the same as the probability that the first digit of a length in feet is 3, 4, or 5. Applying this to all possible measurement scales gives the logarithmic distribution of Benford's law.

3.1.2. Data conformity to Benford's law

After analyzing a dataset using Benford's law, it needs to be tested for compliance. This helps determine the quality of the data, in this case how closely the real data matches the expected distribution. Since a Benford distribution is categorical, many tests can be applied to it:

- Chi-squared goodness of fit

  The *Pearson's chi-squared test of goodness of fit* commonly referred to as the *chi-squared test*[73] is a test which can be applied to data to evaluate it's appropriate for compliance with a theoretical distribution.

  The cumulative chi-squared test statistic χ^2 is formally defined as,

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{8}$$

  where, O_i is the observed value, E_i and is the expected value. Since the counts for n possibilities are approximately normal, they are not considered to be independent. This is because one of the counts can be determined by obtaining the sum of the others. For this reason, the distribution of the chi-square test statistic based on n counts is approximately the chi-square distribution $n - 1$ degrees of freedom[74].

  After obtaining the test statistic for a data set, a null hypothesis H_0 comparison may be performed using a value governed by the expected distribution. A significance level may be chosen (usually, 0.05. Which represents a 5% chance that the

Table 2. Critical Values of the Chi-Square Distribution

df	**Probability (p)**										
	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001
1	0.004	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.64	10.83
2	0.10	0.21	0.45	0.71	1.39	2.41	3.22	4.60	5.99	9.21	13.82
3	0.35	0.58	1.01	1.42	2.37	3.66	4.64	6.25	7.82	11.34	16.27
4	0.71	1.06	1.65	2.20	3.36	4.88	5.99	7.78	9.49	13.28	18.47
5	1.14	1.61	2.34	3.00	4.35	6.06	7.29	9.24	11.07	15.09	20.52
6	1.63	2.20	3.07	3.83	5.35	7.23	8.56	10.64	12.59	16.81	22.46
7	2.17	2.83	3.82	4.67	6.35	8.38	9.80	12.02	14.07	18.48	24.32
8	2.73	3.49	4.59	5.53	7.34	9.52	11.03	13.36	15.51	20.09	26.12
9	3.32	4.17	5.38	6.39	8.34	10.66	12.24	14.68	16.92	21.67	27.88
10	3.94	4.86	6.18	7.27	9.34	11.78	13.44	15.99	18.31	23.21	29.59

variations in the data are due to natural causes), and compared based on the corresponding value in the critical values probability table as seen in table 2.

- Mean absolute deviation

  The mean absolute deviation[76] is a measure of conformity that that ignores the number of records in the data set. A low level of error indicates that the given time series values are a close approximation of the values which were forecast and be deemed reliable. The mean absolute deviation D_{mean} is given by,

$$D_{mean} = \frac{\sum_{i=1}^{K} |x_i - \overline{x}|}{K} \tag{9}$$

where $\overline{x}$ denotes the expected proportion, AP denotes the actual proportion, and K represents the number of bins (which equals 90 for the first two digits).

Simply put, the MAD is the average deviation between the heights of the bins if the distribution were plotted as a histogram and is not dependent on the unit. The higher the MAD, the larger the average difference between the actual and expected proportions. As a result, the MAD compares the conformity of the expected and actual values. Therefore, the data with the lowest MAD is said to have the closest conformity to Benford's Law.

4. Experimentation and Implementation

4.1. Implementation

The system described above was implemented to a limited proof of concept stage. Meaning that not all the functionality described above has been implemented to completion. The implementation was spread between three open source programming languages namely, the Java[77], Groovy and R[78] programming languages. Java was selected as a result of the rich communications APIs available as well as rich community support. Support for WebSockets in Java are described as part of the JSR-356[79] specification. The proof of concept implementation uses an embedded Jetty server[80] which has a built-in implementation of JSR 356[81]. Groovy was selected as a companion JVM language due to it's complete interoperability with Java and less verbose syntax. R was selected because of it's strong focus and support for statistical programming.

```
// Get the class loader used by the system
// Load the jar
URLClassLoader child = new URLClassLoader([jar.toURI().toURL()],
    this.getClass().getClassLoader())
// Get resources contained within the Jar
// and load the designated manifest file
String pluginMetadata = child.getResourceAsStream(PLUGIN_FILE).text
// Deserialize the metadata
Plugin plugin = Utils.deserializePluginMetadata(pluginMetadata)
// Load the class implementing relevant system API
Class<Handler> clazz = (Class<Handler>)
    Class.forName(plugin.handler, true, child)
// Instantiate the class
Handler instance = clazz.newInstance()
// Keep a reference to the instantiated object
handlers.add(instance);
```

Figure 23. *Sample Classloader aimed at loading external *.jar files (Groovy)*

Plugins are shipped as compiled *.jar files which are Java's packaged Zip format[82]. They are then handed off through to the system application layer either through the Classpath or via an accessible location on the file system and are loaded using a ClassLoader.

```
@WebSocket
public class WebSocketHandler {
    // List of nodes maintaining an active connection.
    private static List<Session> connectedNodes =
        new ArrayList<>();
    // Called when connection has been opened with another node
    @OnWebSocketConnect
    public void onConnect(final Session node) throws Exception {
        // Keep reference to connected node
        connectedNodes.add(node)
    }
    // Called when connection has been closed
    @OnWebSocketClose
    public void onClose(final Session node, final int statusCode,
        String reason) {
        //Remove reference to disconnected node
        connectedNodes.remove(node)
    }
    // Called when a message is received from another node
    @OnWebSocketMessage
    public void onMessage(Session node, String message) {
        //Handle message ...
    }
}
```

Figure 24. *Websocket implementation*

The implementation for web sockets in Java is fairly straight forward using Jetty's implementation in the embedded Jetty server. The @Websocket annotation is used to tag a Java Object as being a web socket. During instantiation, method annotations are used to indicate which methods should be called after WebSocket events such as when a connection is opened, closed or when a message is received. Figure 27 shows an implementation of the web sockets used in the system.

Serializing and deserializing in the application is done using Jackson JSON processor library[83]. Jackson was selected because of the maturity of the project as well as the availability of data binding which reduces the need for boilerplate code.

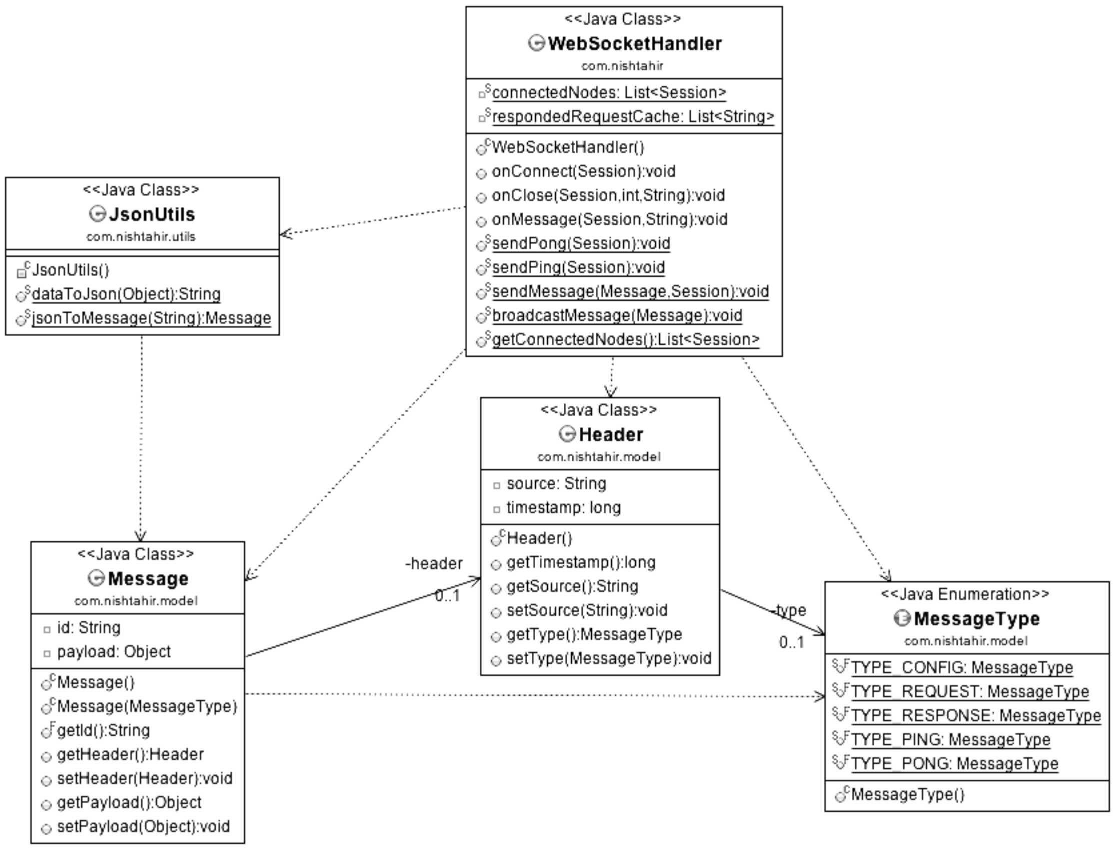


Figure 25. *UML diagram of proof of concept implementation*

Practical use of this library is delegated to a static utility class; in this case JsonUtils, which contains convenience methods for serializing and deserializing to and from various model objects.

```
public static String dataToJson(Object what) {
    ObjectMapper mapper = new ObjectMapper();
    try {
        return mapper.writeValueAsString(what);
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
    return null;
}
```

Figure 26. *Jackson serialization from Object to JSON*

Jackson makes use of an ObjectMapper which implicitly, thus reflectively maps the instance members as well as their values in the class to key-value pairs in the JSON object members[84]. While this is slower than explicit mappings done manually through annotations[85], it works reliably for the proof of concept implementation. When putting together a final implementation however , explicit declaration of mappings should be preferred.

```
        public static Message jsonToMessage(String json) {
         ObjectMapper mapper = new ObjectMapper();
         try {
             return mapper.readValue(json, Message.class);
         } catch (IOException e) {
             e.printStackTrace();
         }
         return null;
     }
```

Figure 27. *Jackson deserialization from JSON to Message*

Similarly, deserialization is also done through the object mapper which instantiates a new object of the given type and reflectively maps the keys and values in the JSON object to instance members of the class.

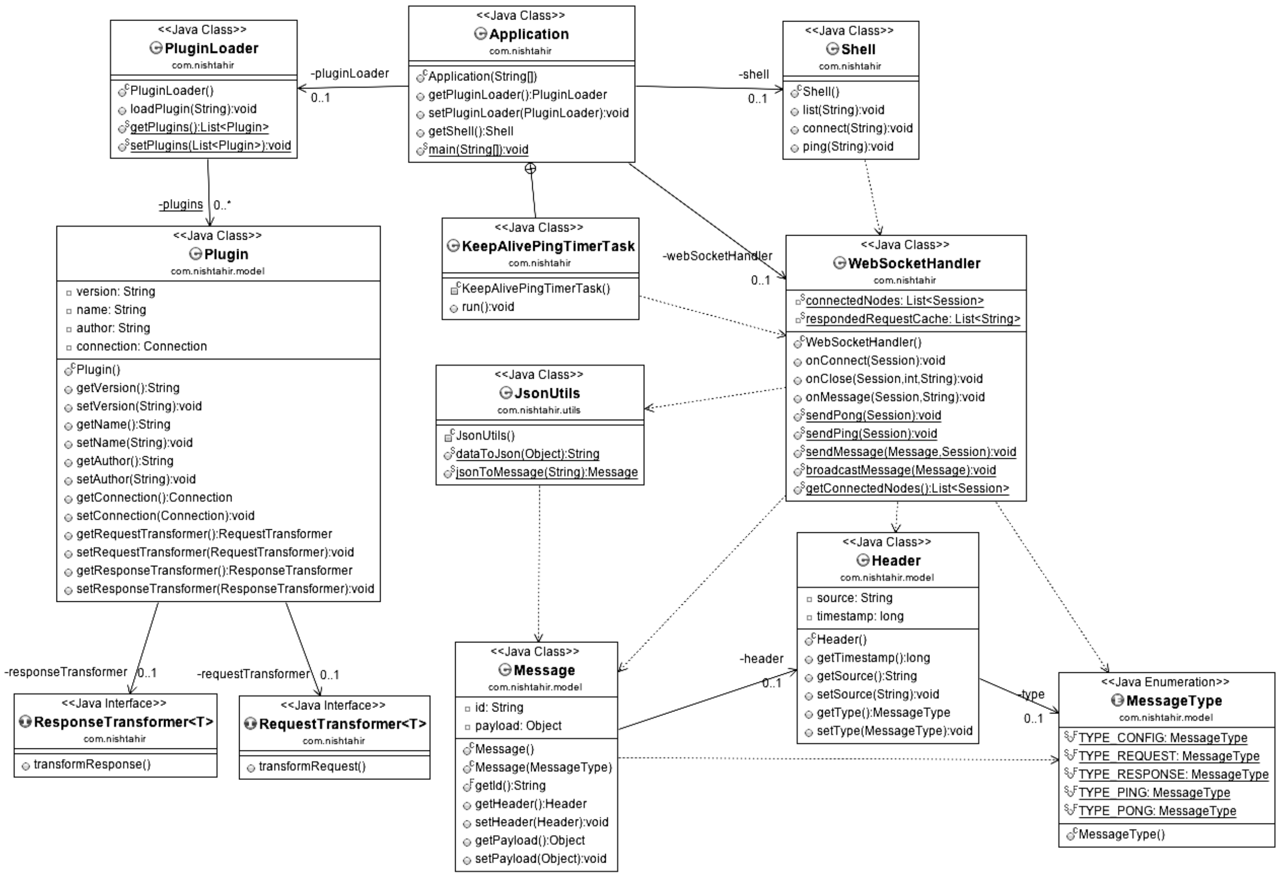


Figure 28. *UML diagram of proof of concept implementation*

The system structure is described in its entirety in figure 28 as a UML diagram. It shows much of what has been described above as it has been implemented as a proof of concept system. An instance of the embedded Jetty server is initialized in the Application class which uses an instance of the PluginLoader to load plugins. It keeps a reference to the interactive Shell which allows users to enter commands. From this, the user may open new connections to other nodes and perform queries.

4.2. Estonian Water Information System

The Estonian Environmental information Center (EEIC) has a large number of databases tasked with the sole purpose of publishing open data. However these databases have multiple different architectures and structures. To mitigate this problem the Estonian Water Information System (EWIS) was planned and is already being developed using these databases. The EWIS database already contains some interesting public information about rivers in Estonia, some of which span over 80 years. For this experiment, we are interested in two water stations, Keila-019 located along the Keila river in Northern Estonia and Kasar-017 located along the Kasari river in western Estonia.

This experiment considers a single metric which is the flow rate q measured in m^3/s.

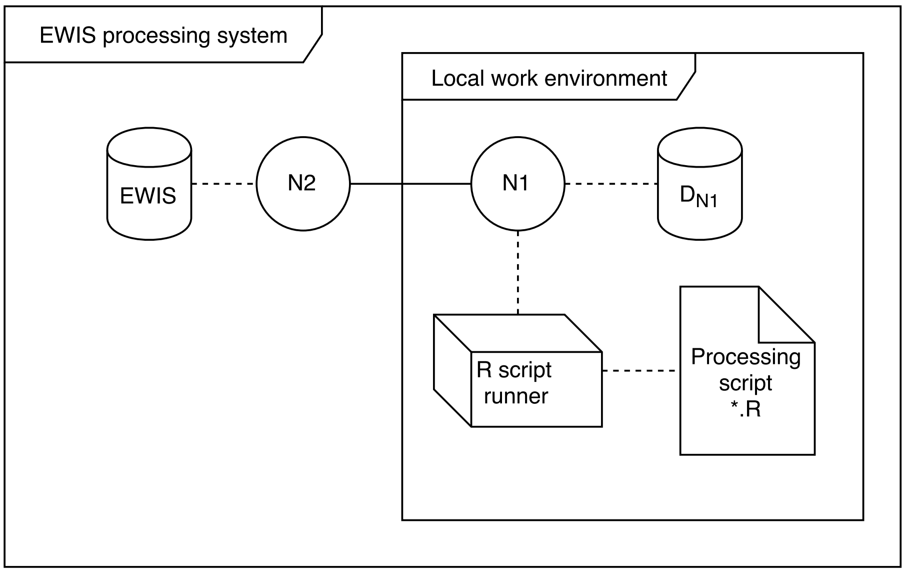


Figure 29. *Structure of EWIS test environment*

The structure of the experimental system is illustrated in figure 29. The EWIS database was hosted on node $N2$ and requests were invoked from $N1$ and the response was processed using an R script. The goal of this experiment was to determine the amount of uncertainty in the data by running it against Benford's law. The data divided into 10-year increments ranging from 1923 to 2010.

The data processing is run against the R script as shown in Appendix 4. The output of the script are plots containing a histogram showing the first digit distribution and a line plot indicating the expected distribution. Also output is the χ^2 goodness of fit for the plot. Based on the χ^2 critical values table as given in table 2, the upper critical value for $\alpha = 0.05$ is 15.51. The results of this experiment yeilded the following results:

Table 3. Keila-019 vs Kasar-017 χ^2 against Benfords law

Period	Keila-019 (χ^2)	Kasar-017 (χ^2)
1923 - 1929	379.456	288.679
1930 - 1939	653.713	98.146
1940 - 1949	541.710	66.723
1950 - 1959	250.969	63.049
1960 - 1969	220.100	50.215
1970 - 1979	256.574	139.872
1980 - 1989	152.562	110.114
1990 - 1999	51.298	56.451
2000 - 2009	230.114	104.853
2010 - 2012	48.199	28.015

These results show that while both datasets are far from within tolerance, Keila-019 is significantly worse off than Kasar-017. However, results from Keila-019 has been improving over the years while Kasar-017 has remained relatively constant. To get a better perspective of the level of variation within the data, it's possible to make a scatter plot of the results with a trend line superimposed over the plot. Another visualization, the confidence region is also represented to show the margin of error in which the trend line may have.

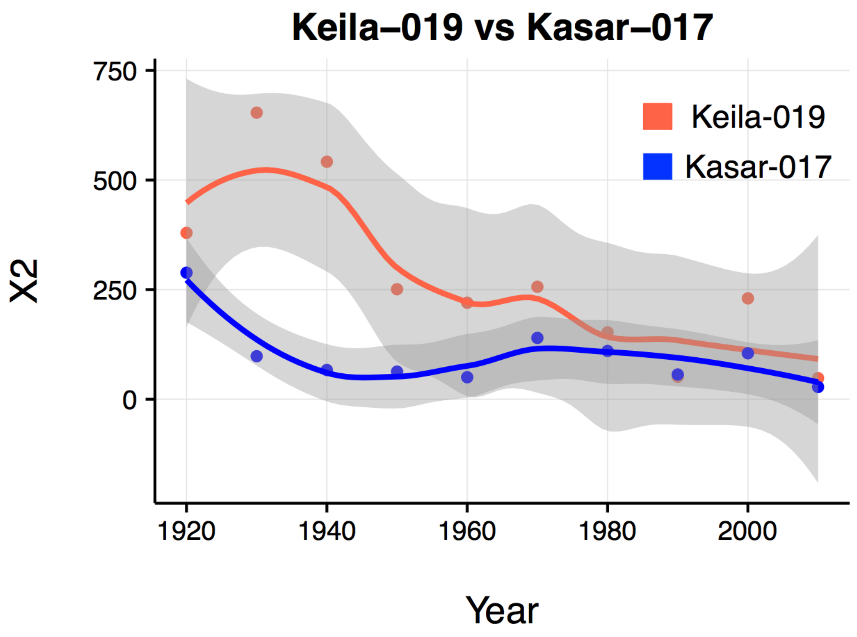


Figure 30. *Smooth plot with confidence region for Keila019 vs Kasar017(Closer to 15.51 is better).*

Figure 30 shows this visualization with the given data set. Here a larger confidence region shows that there is a lot of variation within the scatter plot leaving a wider margin for error in the trend line representing Keila-019's χ^2 results. This is much smaller in Kasar-017 meaning that there is a smaller margin for error. However what is also clear is that there is certainly a trend of Keila-019's results moving closer to that of Kasar-017.

These results show that there is a level of uncertainty in the datasets. However, the results do not provide any insight into the cause of this uncertainty; meaning that one can only speculate as to the cause going by the data alone. The lack of conformity to Benford's law may have been caused by data processing which takes place after data collection or simply averaging which is done to reduce the amount of data which needs to be stored or transferred for processing. The results of these experiments show some promising results which can be paired with other statistical analysis methods designed for this kind of hydrological data such as the Pearson type III[86] function or more general functions such as the product-moment correlation[87] for more conclusive results.

4.3. Prospects

The experiment undertaken by analyzing the content of the EWIS database highlights this platform's significance and possible application in open data unification as well as the benefits of having a modular framework that can be customized by the user to cater to their data processing needs. However, these only scratch the surface regarding its applications. Some other potential uses are discussed.

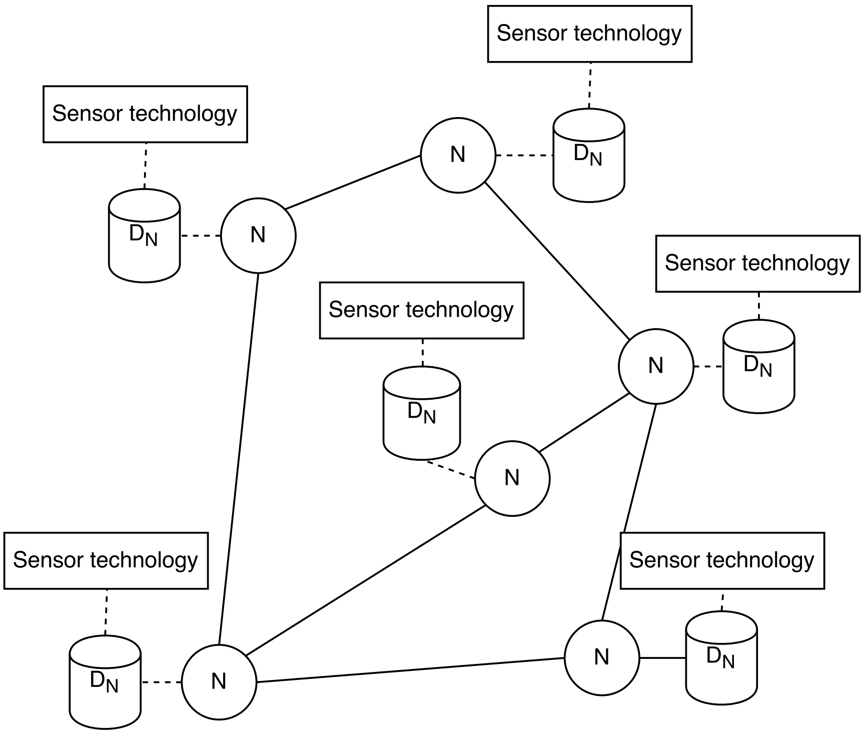


Figure 31. *Open data sensor network.*

By combining the distributed nature of the system with sensor technology, it's possible to create a crowdsourced data publishing environment where data is made available for processing in real time. The value of crowdsourcing sensor data has already been validated through various studies[88] and has proven to be a low-cost way of obtaining large amounts of research information. The advantage that this system brings to the table is that the type of sensor technology that can be implemented is not limited to the base functionality of the system.

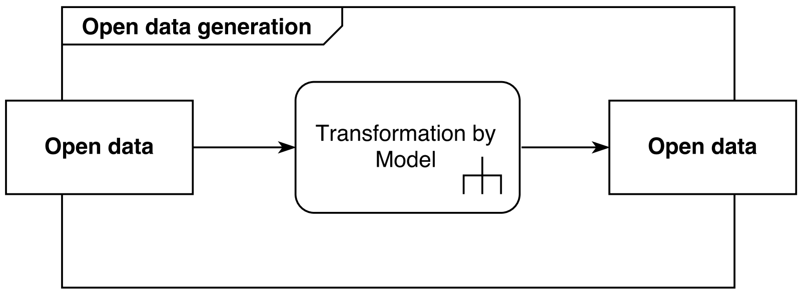


Figure 32. *Open data generation concept[58]*

Since the capabilities of plugins are not limited by APIs that they implement, it means that they may also generate open data. This means that open data collected over the network can be processed using models and output new open data which can also be distributed over the network similar to the concepts used in ROOMA[58].

These use cases only lightly touch on the full capability of the system. I anticipate that its real value will most likely be explored once in the hands of the community. With the separated data access capability and extensible application framework, one can only imagine the kind of use that the community will find in this system.

Conclusion

This thesis has proposed and described a holistic platform for open data linking, data row aggregation and processing. The system works by establishing a highly scalable P2P network, a robust application framework and has proposed multiple techniques for addressing various problems associated with scalability, query abstraction, and data processing. These proposals were validated and tested through the development of a proof of concept system on a combination of open-source technologies based on Java technologies including the JVM platform, the Java, and Groovy programming languages. Experiments were also performed using the EWIS databases to validate the extensibility capabilities of the system as a platform for data analytics in the form of an application targeted at the system implemented using the R programming language for statistical computing which yielded useful positive results.

As a platform for data analysis and computing, it brings together a lot of tools that have been developed towards modern commercial software development. While it's impossible to envision every possible use for this platform, it strives to democratize open data; making it easy and accessible for anyone to collect and host their own data.

For further research into this system, should be focused on standardizing the various components of the system in order to create a protocol specification that can be used as a guide to implementing support on other systems. This standard should cover all components of the system including

1. The communication mechanism addressing things such as the available types of messages that are exchanged by nodes. For example, the Ping and Pong routine[89]; is used to check whether a member of the network is listening for communication.
2. Standardizing the Data source query specification which includes the separated query DSL. It must be robust enough to provide fine granularity in the queries that the user can perform. A good reference for this would be the current SQL specification[90].

Other areas that would benefit research would be in testing, fault tolerance of the system. This should be focused on to ensure that the network has an adequate means recovering if critical nodes are taken down as well as guarantee responsiveness under specific timing constraints. Security also needs to be addressed, and while it was touched on lightly, was not the focus of this paper. As previously described, nodes must currently implicitly trust other nodes to forward requests and other data reliably. However, this is hardly the case in real life. Nodes that bear malicious intent could easily change the content of request and responses.

I see a lot of room for maturity and growth in this system, not only because it fills a need in the open data space but because of the amount of room for expansion that it can provide as a platform for publishing, data aggregation as well as a platform for analysis.

References

[1] MIT Technology Review. (2016). Big Data: Creating the Power to Move Heaven and Earth. [online] Available at: https://www.technologyreview.com/s/530371/big-data-creating-the-power-to-move-heaven-and-earth/ [Accessed 24 May 2016].

[2] Ranganathan, P., 2011. The Data Explosion.

[3] Press, G. (2016). Forbes Welcome. [online] Forbes.com. Available at: http://www.forbes.com/sites/gilpress/2015/04/30/the-supply-and-demand-of-data-scientists-what-the-surveys-say/#25c53ad0205e [Accessed 24 May 2016].

[4] Group, O. (2016). Open Definition 2.1 - Open Definition - Defining Open in Open Data, Open Content and Open Knowledge. [online] Opendefinition.org. Available at: http://opendefinition.org/od/2.1/en/ [Accessed 24 May 2016].

[5] Opendatahandbook.org. (2016). What is Open Data?. [online] Available at: http://opendatahandbook.org/guide/en/what-is-open-data/ [Accessed 24 May 2016].

[6] Data.gov. (2016). About Data.gov - Data.gov. [online] Available at: https://www.data.gov/about [Accessed 24 May 2016].

[7] Haslhofer, B. and Isaac, A., 2011, September. data. europeana. eu: The europeana linked open data pilot. In International Conference on Dublin Core and Metadata Applications (pp. 94-104).

[8] Smith, J., Bernstein, P., Dayal, U., Goodman, N., Landers, T., Lin, K. and Wong, E. (1981). Multibase. Proceedings of the May 4-7, 1981, national computer conference on - AFIPS '81.

[9] Kementsietsidis, A., Arenas, M. and Miller, R.J., 2003, June. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (pp. 325-336). ACM.

[10] Bizer, C., Heath, T., Idehen, K. and Berners-Lee, T. (2008). Linked data on the web (LDOW2008). Proceeding of the 17th international conference on World Wide Web - WWW '08.

[11] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R. and Ives, Z. (2007). DBpedia: A Nucleus for a Web of Open Data. The Semantic Web, pp.722-735.

[12] Chen, C. and Ma, H. (2007). The uncertainty effects of design flow on water quality management. Environmental Monitoring and Assessment, 144(1-3), pp.81-91.

[13] Ics.uci.edu. (2016). Fielding Dissertation: CHAPTER 2: Network-based Application Architectures. [online] Available at: http://www.ics.uci.edu/˜fielding/pubs/dissertation/net_app_arch.htm [Accessed 21 May 2016].

[14] Furht, B., Phoenix, C., Yin, J. and Aganovic, Z., 2000. Internet architectures for application service providers. Handbook of Internet computing, p.67.

[15] Elbert, B. and Martyna, B. (1994). Client/server computing. Boston: Artech House.

[16] Bertocco, M., Ferraris, F., Offelli, C. and Parvis, M. (1998). A client-server architecture for distributed measurement systems. IEEE Trans. Instrum. Meas., 47(5), pp.1143-1148.

[17] Ics.uci.edu. (2016). Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST). [online] Available at: http://www.ics.uci.edu/˜fielding/pubs/dissertation/rest_arch_style.htm [Accessed 21 May 2016].

[18] Milojicic, D.S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S. and Xu, Z., 2002. Peer-to-peer computing.

[19] Schollmeier, R., 2001, August. [16] A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In p2p (p. 0101). IEEE.

[20] Bandara, H. and Jayasumana, A. (2012). Collaborative applications over peer-to-peer systems–challenges and solutions. Peer-to-Peer Networking and Applications, 6(3), pp.257-276.

[21] Vu, Q., Lupu, M. and Ooi, B. (2010). Peer-to-peer computing. Heidelberg: Springer, pp.2 - 4.

[22] Setiathome.berkeley.edu. (2016). About SETI@home. [online] Available at: http://setiathome.berkeley.edu/sah_about.php [Accessed 21 May 2016].

[23] Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M. and Werthimer, D., 2002. SETI@ home: an experiment in public-resource computing. Communications of the ACM, 45(11), pp.56-61.

[24] Msdn.microsoft.com. (2016). Benefits of Peer Networking (Windows). [online] Available at: https://msdn.microsoft.com/en-us/library/windows/desktop/dd433180(v=vs.85).aspx [Accessed 21 May 2016].

[25] Koegel Buford, J., Yu, H. and Lua, E. (2009). P2P networking and applications. Amsterdam: Elsevier/Morgan Kaufmann, pp.31 - 35.

[26] Beineke, L. and Wilson, R. (1997). Graph connections. Oxford: Clarendon Press, pp.53 - 60.

[27] Mathworld.wolfram.com. (2016). Directed Graph – from Wolfram MathWorld. [online] Available at: http://mathworld.wolfram.com/DirectedGraph.html [Accessed 21 May 2016].

[28] Mathworld.wolfram.com. (2016). Undirected Graph – from Wolfram MathWorld. [online] Available at: http://mathworld.wolfram.com/UndirectedGraph.html [Accessed 21 May 2016].

[29] Preiss, B. (2016). Sparse vs. Dense Graphs. [online] Brpreiss.com. Available at: http://www.brpreiss.com/books/opus4/html/page534.html [Accessed 21 May 2016].

[30] Mathworld.wolfram.com. (2016). Graph Diameter – from Wolfram MathWorld. [online] Available at: http://mathworld.wolfram.com/GraphDiameter.html [Accessed 21 May 2016].

[31] Myers, A. (2016). Graph traversals. [online] Cs.cornell.edu. Available at: http://www.cs.cornell.edu/courses/cs2112/2012sp/lectures/lec24/lec24-12sp.html [Accessed 21 May 2016].

[32] Rodriguez, M.A. and Neubauer, P., 2010. The graph traversal pattern. arXiv preprint arXiv:1004.1001.

[33] Dowek, G. (2009). Principles of programming languages. London: Springer, p.143.

[34] Dowek, G. (2009). Principles of programming languages. London: Springer, p.145.

[35] Taylor, I. (2005). From P2P to Web services and grids. London: Springer. p.241

[36] Tools.ietf.org. (2016). RFC 7230 - Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. [online] Available at: https://tools.ietf.org/html/rfc7230 [Accessed 21 May 2016].

[37] Hanson, J. (2014). What is HTTP Long Polling? | PubNub. [online] PubNub. Available at: https://www.pubnub.com/blog/2014-12-01-http-long-polling/ [Accessed 15 May 2016].

[38] Hypertext transfer protocol (HTTP) keep-alive header (2010) [online] Available at: https://tools.ietf.org/id/draft-thomson-hybi-http-timeout-01.html [Accessed 15 May 2016].

[39] Docs.oracle.com. (2016). What Is a Socket? (The Java^{TM} Tutorials > Custom Networking > All About Sockets). [online] Available at: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html [Accessed 21 May 2016].

[40] Oracle.com. (2016). Lesson 1: Socket Communications. [online] Available at: http://www.oracle.com/technetwork/java/socket-140484.html [Accessed 21 May 2016].

[41] Lombardi, A. (2015). WebSocket: Lightweight Client-Server Communications. p.1

[42] Tools.ietf.org. (2016). RFC 6455 - The WebSocket protocol. [online] Available at: https://tools.ietf.org/html/rfc6455 [Accessed 1 May 2016].

[43] Websocket.org. (2016). About HTML5 WebSocket - Powered by Kaazing. [online] Available at: http://www.websocket.org/aboutwebsocket.html [Accessed 1 May 2016].

[44] Kelaskar, M., Matossian, V., Mehra, P., Paul, D. and Parashar, M. (n.d.). A Study of Discovery Mechanisms for Peer-to-Peer Applications. 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02).

[45] Ripeanu, M., 2001, August. Peer-to-peer architecture case study: Gnutella network. In Peer-to-Peer Computing, 2001. Proceedings. First International Conference on (pp. 99-100). IEEE.

[46] Rfc-gnutella.sourceforge.net. (2016). Gnutella Protocol Development. [online] Available at: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html [Accessed 22 May 2016].

[47] Bo, J. (2011). Flooding-Based Resource Locating in Peer-to-Peer Networks. Lecture Notes in Electrical Engineering, pp.671-678.

[48] En.bitcoin.it. (2016). Satoshi Client Node Discovery - Bitcoin Wiki. [online] Available at: https://en.bitcoin.it/wiki/Satoshi_Client_Node_Discovery [Accessed 7 May 2016].

[49] Ibm.com. (2016). IBM Knowledge Center. [online] Available at: http://www.ibm.com/support/knowledgecenter/SSPK3V_6.3.0/com.ibm.swg.im.soliddb.sql.doc/doc/tables.rows.and.columns.html [Accessed 22 May 2016].

[50] Darwen, H. (2009). An introduction to relational database theory. [Place of publication not identified]: Ventus Publishing.

[51] MongoDB. (2016). NoSQL Databases Explained. [online] Available at: https://www.mongodb.com/nosql-explained [Accessed 22 May 2016].

[52] W3.org. (2015). Model for Tabular Data and Metadata on the Web. [online] Available at: https://www.w3.org/TR/tabular-data-model/ [Accessed 21 May 2016].

[53] Arvin, T. (2016). Comparison of different SQL implementations. [online] Troels.arvin.dk. Available at: http://troels.arvin.dk/db/rdbms/ [Accessed 22 May 2016].

[54] Bartholomew, D., 2010. SQL vs. NoSQL. Linux Journal, 2010(195), p.4.

[55] Sametinger, J. (1997). Software Engineering with Reusable Components. Berlin, Heidelberg: Springer Berlin Heidelberg, p.24.

[56] Hibernate.org. (2016). Hibernate ORM - Hibernate ORM. [online] Available at: http://hibernate.org/orm/ [Accessed 22 May 2016].

[57] Msdn.microsoft.com. (2016). LINQ to SQL. [online] Available at: https://msdn.microsoft.com/en-us/library/bb386976(v=vs.110).aspx [Accessed 22 May 2016].

[58] Viies, V., Aigro, J., Kinks, H., Kullamaa, R., Ozolit, M.O (2014). ROOMA BRINGS VICTORY. [online] Available at: http://www.tud.ttu.ee/im/Vladimir.Viies/materials/GAG/EWIS%20-%20Italy5.doc [Accessed 4 May 2016].

[59] Viies, V.; Ennet, P.; Aigro, J.; Kinks, H.; Kullamaa, R.; Ozolit, O. M. & Salula, A. (2012), Processing Multiple Databases in the Estonian Water Information System., in Albertas Caplinskas; Gintautas Dzemyda; Audrone Lupeikiene & Olegas Vasilecas, ed., 'DB&Local Proceedings' , CEUR-WS.org, , pp. 29-36 .

[60] Mens, T. (2012). On the Complexity of Software Systems. Computer, 45(8), pp.79-81.

[61] Abbes, M., Khomh, F., Gueheneuc, Y. and Antoniol, G. (2011). An Empirical Study of the Impact of Two Antipatterns, Blob and Spaghetti Code, on Program Comprehension. 2011 15th European Conference on Software Maintenance and Reengineering.

[62] The Benefits of Modular Programming. (2007). 1st ed. [ebook] Sun Microsystems. Available at: https://netbeans.org/project_downloads/usersguide/rcp-book-ch2.pdf [Accessed 10 May 2016].

[63] Tools.ietf.org. (2016). RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace. [online] Available at: https://tools.ietf.org/html/rfc4122 [Accessed 17 May 2016].

[64] Arnedo-Moreno, J. and Herrera-Joancomartı, J., Persistent interoperable security for JXTA advertisements.

[65] Hericko, M., Juric, M.B., Rozman, I., Beloglavec, S. and Zivkovic, A., 2003. Object serialization analysis and comparison in java and. net. ACM Sigplan Notices, 38(8), pp.44-54.

[66] Bhatti, N., Hassan, W., McClatchey, R., Martin, P., Kovacs, Z., Le Goff, J.M., Stockinger, H., Willers, I. and Le, J.M., 2000. Object Serialization and Deserialization Using XML.

[67] De Jong, S.P., Kakivaya, G.K.R. and Roxe, J.L., 2005. Architecture and method for serialization and deserialization of objects. U.S. Patent 6,928,488.

[68] W3.org. (2016). Extensible Markup Language (XML) 1.0 (Fifth Edition). [online] Available at: https://www.w3.org/TR/REC-xml/ [Accessed 22 May 2016].

[69] Yaml.org. (2016). YAML^{TM} Specification Index. [online] Available at: http://yaml.org/spec/ [Accessed 22 May 2016].

[70] Tools.ietf.org. (2016). RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format. [online] Available at: https://tools.ietf.org/html/rfc7159.html [Accessed 22 May 2016].

[71] Pike, D.P., (2008). Testing for the benford property. Working paper.

[72] Diekmann, A. (2007). Not the First Digit! Using Benford's Law to Detect Fraudulent Scientif ic Data. Journal of Applied Statistics, 34(3), pp.321-329.

[73] Jstor.org. (2016). Karl Pearson and the Chi-Squared Test on JSTOR. [online] Available at: http://www.jstor.org/stable/1402731 [Accessed 1 May 2016].

[74] Stat.yale.edu. (2016). Chi-Square Goodness of Fit Test. [online] Available at: http://www.stat.yale.edu/Courses/1997-98/101/chigf.htm [Accessed 15 May 2016].

[75] Itl.nist.gov. (2016). Critical Values of the Chi-Square Distribution. [online] Available at: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3674.htm [Accessed 15 May 2016].

[76] Benford's Law: Applications for Forensic Accounting, Auditing, and Fraud Detection. (2012). New York: John Wiley & Sons, p.34.

[77] Docs.oracle.com. (2016). Java SE Specifications. [online] Available at: https://docs.oracle.com/javase/specs/ [Accessed 22 May 2016].

[78] Team, R.C., 2000. R Language Definition. Available from CRAN sites.

[79] Oracle.com. (2016). JSR 356, Java API for WebSocket. [online] Available at: http://www.oracle.com/technetwork/articles/java/jsr356-1937161.html [Accessed 22 May 2016].

[80] Eclipse.org. (2016). Embedding Jetty. [online] Available at: http://www.eclipse.org/jetty/documentation/current/embedding-jetty.html [Accessed 22 May 2016].

[81] Eclipse.org. (2016). Chapter29.Jetty Websocket API. [online] Available at: http://www.eclipse.org/jetty/documentation/current/websocket-jetty.html [Accessed 2 May 2016].

[82] Docs.oracle.com. (2016). Using JAR Files: The Basics (The Java^{TM} Tutorials > Deployment > Packaging Programs in JAR Files). [online] Available at: https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html [Accessed 22 May 2016].

[83] Wiki.fasterxml.com. (2016). JacksonHome - FasterXML Wiki. [online] Available at: http://wiki.fasterxml.com/JacksonHome [Accessed 24 May 2016].

[84] Wiki.fasterxml.com. (2016). JacksonInFiveMinutes - FasterXML Wiki. [online] Available at: http://wiki.fasterxml.com/JacksonInFiveMinutes [Accessed 24 May 2016].

[85] Wiki.fasterxml.com. (2016). JacksonAnnotations - FasterXML Wiki. [online] Available at: http://wiki.fasterxml.com/JacksonAnnotations [Accessed 24 May 2016].

[86] Mathworld.wolfram.com. (2016). Pearson Type III Distribution – from Wolfram MathWorld. [online] Available at: http://mathworld.wolfram.com/PearsonTypeIIIDistribution.html [Accessed 24 May 2016].

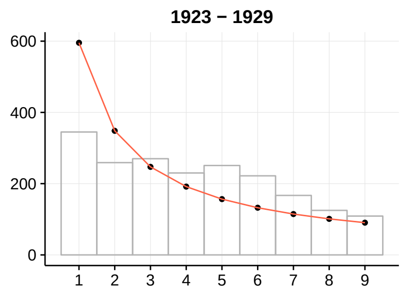
[87] Statistics.laerd.com. (2016). Pearson Product-Moment Correlation [online] Available at: https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php [Accessed 24 May 2016].

[88] Myshake.berkeley.edu. (2016). MyShake. [online] Available at: http://myshake.berkeley.edu/ [Accessed 19 May 2016].

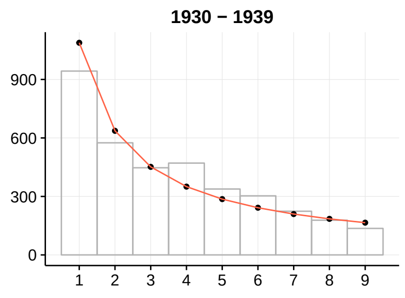
[89] Gray, J. and Reuter, A. (1993). Transaction processing. San Mateo, Calif.: Morgan Kaufmann Publishers.

[90] ISO. (2016). ISO/IEC 9075-1:2008 - Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). [online] Available at: http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498 [Accessed 26 May 2016].

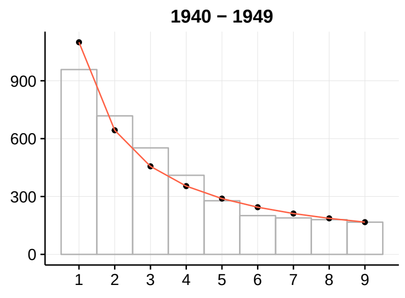
Appendix 1



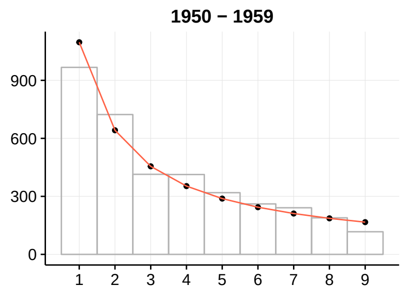
(a) $\chi^2 = 288.697$



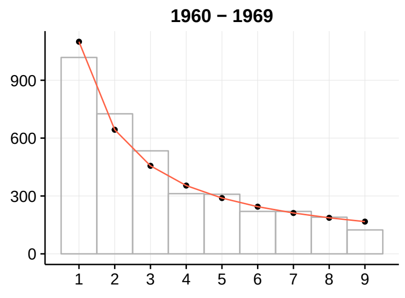
(b) $\chi^2 = 98.146$



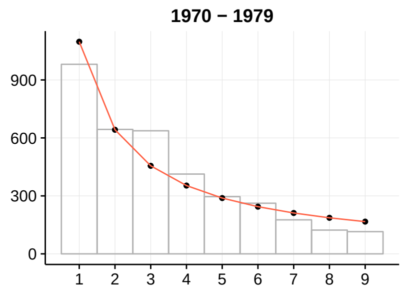
(c) $\chi^2 = 66.723$



(d) $\chi^2 = 63.049$

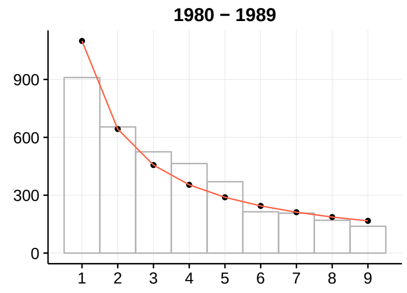


(e) $\chi^2 = 50.215$

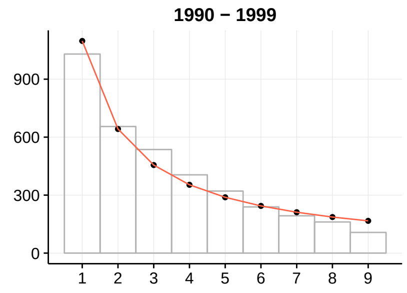


(f) $\chi^2 = 139.872$

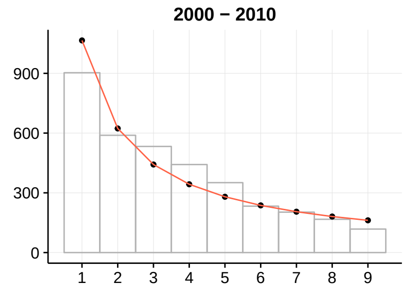
Figure 33. Kasar-017 flow rate results



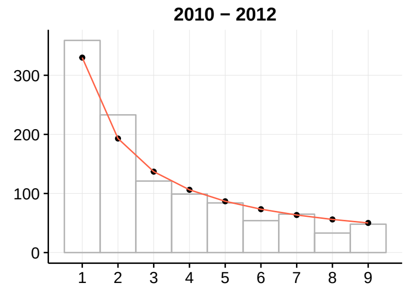
(g) $\chi^2 = 110.114$



(h) $\chi^2 = 56.451$



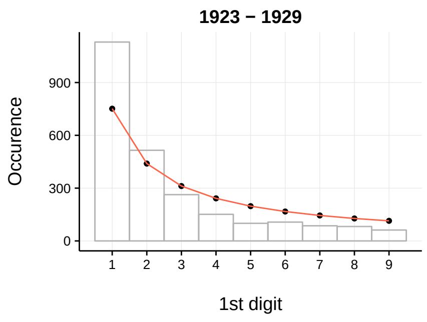
(i) $\chi^2 = 104.853$



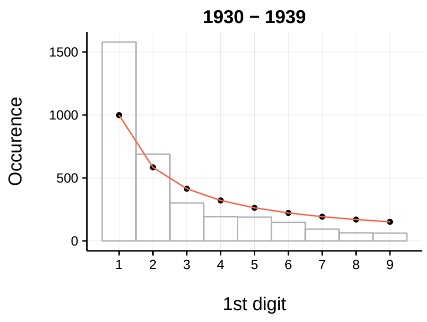
(j) $\chi^2 = 28.015$

Figure 34. Kasar-017 flow rate results, contd.

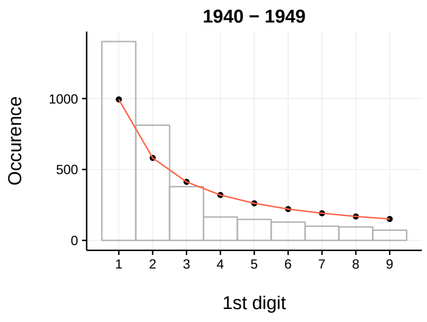
Appendix 2



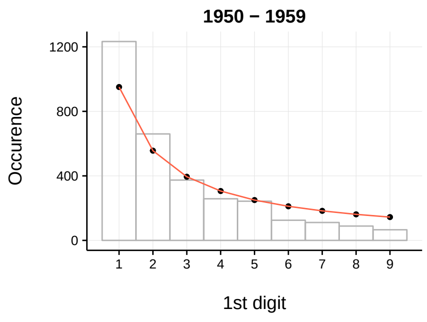
(a) $\chi^2 = 379.465$



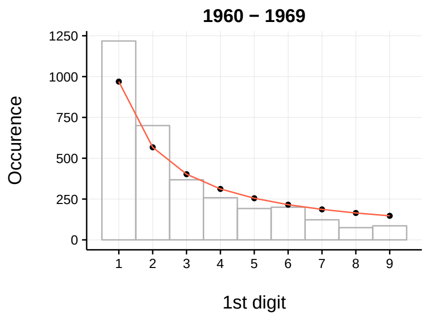
(b) $\chi^2 = 653.713$



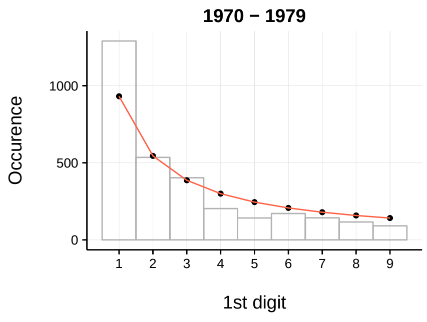
(c) $\chi^2 = 541.710$



(d) $\chi^2 = 250.969$

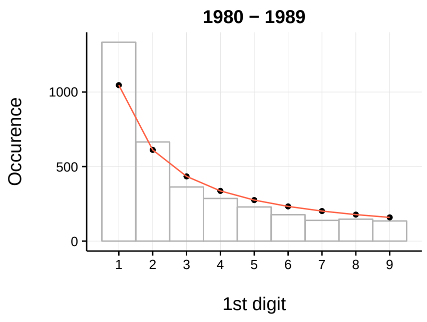


(e) $\chi^2 = 220.100$

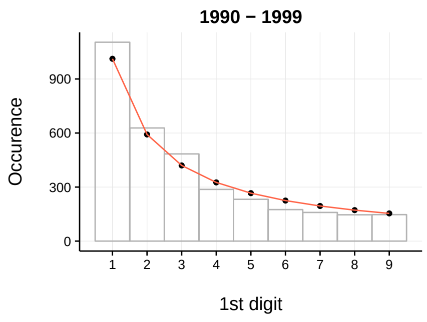


(f) $\chi^2 = 256.574$

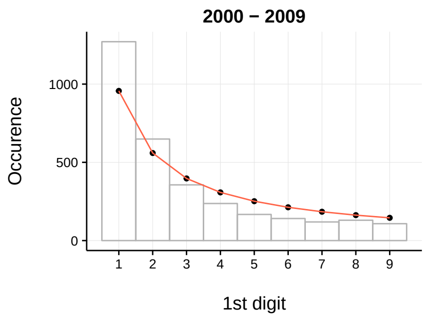
Figure 35. Keila-019 flow rate results



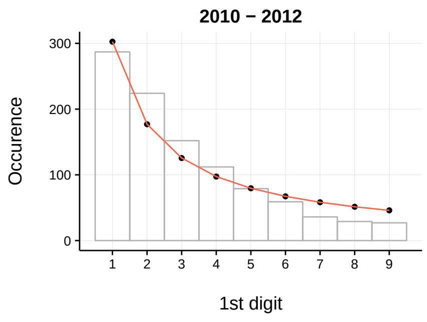
(g) $\chi^2 = 152.562$



(h) $\chi^2 = 51.298$



(i) $\chi^2 = 230.114$



(j) $\chi^2 = 48.199$

Figure 36. Keila-019 flow rate results, contd.

Appendix 3

```r
#' @title Benfords law analysis script
#' @description Parses a given csv file and analyzes its
#'              content against Benford's law.
#'              Outputs a *.pdf file with the results
#' @author Nish Tahir <nishtahir@outlook.com>
#'
if (!require("pacman"))
  install.packages("pacman")
pacman::p_load("ggplot2", "gridExtra", "cowplot")
#' Calculates the cumulative test statistic
#' @param real
#' @param expected
#' @return culative test statistic for given values
calculateTestStatistic <- function(real, expected) {
  return (((real - expected) ^ 2) / expected)
}
#' Add together two numbers.
#' @param listReal list containing the real values
#' @param listTheo A number.
#' @return The sum of \code{x} and \code{y}.
getCumulativeTestStatistic <- function(listReal, listTheo) {
  sum = 0
  for (i in 1:length(listReal)) {
    sum = sum + calculateTestStatistic(listReal[i], listTheo[i])
  }
  sum
}
```

Figure 37. *Benfords law R script - 1*

```
#' Add together two numbers.
#' @param listReal list containing the real values
#' @param listTheo A number.
#' @return The sum of \code{x} and \code{y}.
getCumulativeTestStatistic <- function(listReal, listTheo) {
  sum = 0
  for (i in 1:length(listReal)) {
    sum = sum + calculateTestStatistic(listReal[i], listTheo[i])
  }
  sum
}
#' Calculates predicted value distribution based on
#' benfords law
#' @param k length of vector
#' @return
getBenfordPredictionList <- function(k) {
  list <- c()
  for (i in 1:9) {
    list[i] = (theoretical_percentages$first[i + 1] / 100) * k
  }
  list
}
#' Extract digit from numeric value
#'
#' @param value Value to operate on
#' @param index Index to extract
#' @return
#' @example
#' example(1234, 1)
extractDigit <- function(value, index) {
  val <- gsub("[^0-9]", "", as.character(value))
  val <- strsplit(val, '')
  as.numeric(val[[1]][index])
}
```

Figure 38. *Benfords law R script - 2*

```
#’ Plots histogram with given list and saves it
#’ to a *.pdf file matching the given filename
#’ @param list
#’ @param fileName
#’ @param theo
#’ @param statistic
plotAndSaveHistogram <- function (list, fileName, theo, stastic) {
  par(bg = ’white’)
  image <- ggplot() +
    aes(list) +
    geom_histogram(binwidth = 1,
                   color = "gray70",
                   fill = NA) +
    geom_point(aes(x = as.numeric(c(1:9)), y = theo), color = "black")
    geom_line(aes(x = as.numeric(c(1:9)), y = theo), color = "tomato1")
    scale_x_continuous(breaks = c(1:9)) +
    background_grid(major = "xy", minor = "none") +
    labs(title = strsplit(basename(fileName), "\\.")[[1]],
         x = "\n1st digit",
         y = "Occurence\n") +
    theme(
      panel.background = element_rect(fill = ’white’, color = ’black’),
      panel.grid = element_line(color = ’black’),
      plot.margin = unit(c(1, 1, 1, 1), "cm"),
      title = element_text(size = 18),
      axis.text = element_text(size = 10),
      axis.title = element_text(size = 14)
    )
  save_plot(
    file = paste(fileName, "_", ".pdf"),
    plot = image,
    base_aspect_ratio = 1.3
  )
}
```

Figure 39. *Benfords law R script - 3*

```
#' Extract digit from numeric value
#'
#' @param value Value to operate on
#' @param index Index to extract
#' @return
#' @example
performBenfordAnalysis <- function(fileName) {
  values <- read.csv(file = fileName, header = F, sep = ",")
  #init list
  list <- c()
  for (i in values$V4) {
    val <- extractDigit(i, 1)
    if (val != 0) {
      list[length(list) + 1] <- val
    }
  }
  list_theoretical <- getBenfordPredictionList(length(list))
  plotAndSaveHistogram(
    list,
    fileName,
    list_theoretical,
    getCumulativeTestStatistic(as.vector(table(list)),
    list_theoretical)
  )
}
# Theoretical values for first digit
theoretical_percentages <- data.frame(
  c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9),
  c(NA, 30.103, 17.609, 12.494, 9.691, 7.918,
    6.695, 5.799, 5.115, 4.576
  )
)
# Set column names for data frame
colnames(theoretical_percentages) <- c("index", "first")
```

Figure 40. *Benfords law R script - 4*