# Terabyte Threat Analysis
# Requirements Specification

Team Lima
For Paul Reid on behalf of BT

12th February, 2013

**Abstract**

This report is the result of a single client meeting between Team Lima, a group of students from the University of Cambridge Computer Laboratory, and Paul Reid, a representative of British Telecom (BT), pertaining to a software system to analyse threats and data flows across the BT network. It contains a list of technical requirements that the final software system is expected to conform to, along with plans of system implementation and testing criteria.

## Contents

# 1 Introduction

This section outlines the aims and intentions of this requirements specification, and its usefulness and relevance in the guidance of this software project.

## 1.1 Document Scope

This document and all material herein are intended to outline the high-level corporate motivation, justification and requirements behind the development of a near real-time network threat analysis system over massive datasets. It is intended to be suitable for distribution to all parties associated with the project, regardless of their background or degree of technical involvement, if any. It is imperative that the users of any proposed system are fully informed and involved in outlining their demands on the system in order for the project's success to be measured. Nevertheless, it is also crucial for the developers to have a thorough understanding of the design of the proposed system in order to facilitate proper implementation. To that end, we seek to outline the proposed system's impact on business processes which are of relevance to all stakeholders, while also providing an overview of the technical implementation.

## 1.2 Motivation

BT is one of the world's oldest and largest communications providers, responsible for the provision of telephony and data services to a variety of homes, businesses and corporations in the UK and internationally. Its network core consists of a large collection of network routers, devices tasked with directing the flow of data packets for BT's customers between the source and destination computer networks in a timely manner. Each router provides logs and statistical data about the packets of data which have travelled through it, which BT have not utilised up to this point.

As one of BT's largest and most important assets, all network monitoring systems are put in place with the express intent of avoiding all possible downtime with respect to routine equipment failures, an ever-increasing level of cyber attacks, and other day-to-day operational issues. BT's commitment to monitoring plays a role in its efforts to build customer trust in the reliability of its systems, thereby encouraging customers to retain BT as their service provider.

This system aims to be the first indication as to whether the additional data provided by the network routers can be usefully analysed to add value to the current monitoring tools or, alternatively, to answer questions that can be posed about the state of their network for a given period.

## 1.3 System Overview

BT currently makes use of a number of tools, most of which are freely available, to capture, analyse and visualise the operational data about its network. The network infrastructure currently takes samples of packets flowing through the system and creates logfiles in a standard format known as 'nfcapd'. These existing files can be placed in a directory for this new software system to read, a tool such as nfdump[1] converting the data into a more usable comma separated values format. A script will run to process this data into a file system more capable of handling such large amounts of data, namely the Hadoop High Performance File System (HDFS[2]), where it will be accessible to a cluster of processing nodes running Hadoop[3]. This is another open source product capable of abstracting away the distribution of data analysis across many nodes, providing a scalable system whose computational capabilities can be trivially extended as system demands require by simply adding nodes to the cluster. Hadoop is a well-known, industry-standard tool for this purpose.

A number of statistics have been identified through use of sample data from the live network as potential indicators of threats. These statistics have been made the subject of our forthcoming research and implementation into the threat analysis. The nodes in the distributed computing cluster will be programmed with jobs to reduce and aggregate the input data into smaller datasets. The specific jobs assigned to the nodes will remove all extraneous, missing and duplicate information and will exploit the nature of Hadoop to conduct the heavy-lifting to identify interesting events, which will subsequently be stored into a database. The database technology selected, HBase[4], is closely coupled to the Hadoop distributed computing environment to assure performance at scale.

A single Java module will be responsible for parsing these events, selecting only current issues that are detected on the system to pass forward towards the user interface. The data will be placed into a relational database, PostgreSQL[5], directly queryable and modifiable from a python-based web server implementation in Flask[6] through a library called PsycoPG[7]. This will serve pages written in HTML, CSS and Javascript to the end user's browser to display the information from the database.

[1] http://nfdump.sourceforge.net, BSD 3-clause License
[2] http://hadoop.apache.org, Apache v2 License
[3] *ibid.*
[4] http://hbase.apache.org/, Apache v2 License
[5] http://www.postgresql.org/, PostgreSQL License
[6] http://flask.pocoo.org/, BSD 3-clause License
[7] http://initd.org/psycopg/, GNU Lesser General Public License

In addition, Happybase[8] will be used to complete the feedback loop and allow user input to be written back to the underlying HBase database.

This method of implementing a Graphical User Interface (GUI) is noted for being ubiquitous in the delivery of modern software front-end environments and for its widespread support in today's Internet browsers. Using Asynchronous Javascript and XML (AJAX), the interface will be capable of dynamically updating the user's display as new information enters the database. Furthermore, the GUI's use of modern *Web 2.0* technologies complements those already implemented by BT for similar purposes, permitting a future integration of systems should the demand arise.

[8] http://happybase.readthedocs.org/en/latest/, MIT License

# 2 Modular Structure

Several key components have been identified which internally divide the system into workunits capable of being assigned to separate developers. These components, along with the associated design requirements and constraints, are the subject of this section. For diagrams of these sections, please see the appendices.

## 2.1 Importer Tool

This tool will comprise of a single shell script with the sole task of converting .nfcapd binary files into comma separated value text files capable of being loaded into Hadoop, and placing them in a location accessible by the next stage. This is a relatively simple tool, since it uses third-party tools to do the transformation, wrapping them up in a single workflow for simplicity. The file will be run through nfdump, then the result transferred through a named pipe directly into the HDFS. However, this tool will fire periodically through *cron* to poll the directory for new files to copy, so it is possible that two instances of the tool may end up running simultaneously. In this case, a lockfile should be used to ensure only one instance has access to the files, the other exiting with no work done to ensure consistency.

As this module implements behaviour which acts at the file system level, it was considered more appropriate to implement this using a shell script rather than a service written in Java. The number of lines of code would be minimal, making maintenance much easier. In addition, a shell script is very conveniently scheduled as a *cron* task. Having many instances of the script around is unlikely to lead to adverse system performance, unlike an implementation which spawns many instances of a bulky Java Virtual Machine (JVM) at once, on what could be a machine with a specification at the lower-end of the scale.

## 2.2 MapReduce Jobs

The main technology used to handle the large amounts of data provided will be MapReduce, the core algorithm implemented by Hadoop to process and aggregate data at a large scale. It is necessary for each analytic stage to form its own MapReduce job, and thus a list of data that needs to be extracted from the initially provided mass of raw data must be located. For the purpose of identifying threats, a rule-based detection system will be implemented, since most algorithmic models are liable to provide much higher rates of false positives. It has been agreed that the system needs to be careful in how eager it is to report issues so as not to cause operators 'the boy who cried wolf' effect. However, it has also been acknowledged that, given this dataset, it is also entirely feasible to provide system statistics and metrics on the frontend in addition, the details of which can be found below.

The statistics for each router will be calculated separately by a Hadoop job, aggregating on a time period pre-defined to the system. Initially, it is expected that a 'per minute' basis will be acceptable. As it is not expected to be the case that the length of time represented in the file is a multiple of the provided variable, it will be necessary to aggregate the start of each file with the trailing end of the prior file at the same time. The statistics the system will be able to extract will include

- the number of packets that have flown through the router, split into
  - TCP,
  - UDP, and
  - ICMP,
- the number of flows through the router, and
- the total size of data transported through the router.

The threat analysis portion of the MapReduce jobs can be split into two categories:

- Threats detectable by single flow analysis: A single MapReduce job will be sufficient to identify instances of these threats, not requiring data from the network as a whole. Any flows that are likely suspect will be recorded in the database, based upon thresholds set and fine-tuned during operation. The MapReduce jobs created will mirror known flow-based methods (Kim *et al.* 2004[9], pp. 4-5, 7).
- Threats detectable only by multiple flow analysis: Denial of Service (DoS), Distributed Denial of Service (DDoS) attacks, and incoming port scans. These attacks do not only affect one router, nor do they have a common endpoint within the BT network. It is unclear exactly how detectable a port scan will be, since the logfile sample is for 0.1% of all packets, however Kim *et al.* (2004[10]) does suggest that it is still feasible. A MapReduce job can break flows into 10s timeframes, attempting to identify short bursts of excessive activity.

[9] http://dpnm.postech.ac.kr/papers/NOMS/04/security-analysis/camera-ready/attack-analysis-v5-revision.pdf
[10] *ibid.*

There is the potential for this stage of the data pipeline to act as a bottleneck, as the rate at which data is analysed is bound by several factors. Namely,

- the bandwidth of the network backbone for the exchange of the datasets,
- the rate at which HDFS can accept reads,
- the rate at which HBase can perform writes, and
- the number of nodes available (and their associated performance) in the Hadoop cluster for data processing.

## 2.3 HBase Storage Format

HBase will be responsible for storing intermediate data from Hadoop. Two tables are expected to be required.

| Statistics | |
|---|---|
| Column Name | Description |
| routerID | A unique identifier for a given router. |
| timeframe | The start of the given analysis period. |
| flowCount | Number of flows in the time period. |
| packetCount | Number of packets in the time period. |
| totalDataSize | Sum of all the data travelling through the router for the period. |
| TCPCount | Number of TCP packets in the time period. |
| UDPCount | Number of UDP packets in the time period. |
| ICMPCount | Number of ICMP packets in the time period. |

| Threat | |
|---|---|
| Column Name | Description |
| timeProcessed | The time the data is recorded into the database. |
| routerID | A unique identifier for the router in question. |
| attackType | One of the enumeration of possible attack types. |
| startTime | The detected start time of the threat. |
| endTime | The detected end time of the threat. |
| *sourceIP* | Optional data pertaining to the specific threat. |
| *destIP* | |
| *flowCount* | |
| *flowDataAvg* | |
| *flowDataTotal* | |

## 2.4 HBase Cleanup

The architecture of the HBase database means it is capable of scaling to store exceptionally large quantities of data. Nevertheless, it may be naïve to simply allow data to accumulate indefinitely.

Firstly, the relevance of historic data decreases continually with the passage of time. Secondly, it would be impolite to simply fill up the storage medium with unused records, since this is not the only system that is expected to use the Hadoop cluster. Thirdly, a greater amount of data means added effort and issue if attempting to alter records, export and import to a new system, or simply change the existing schema.

This tool is expected to stand separately from the main dataflow of the project. To avoid performance issues during levels of extensive data processing, it will be configured to run as a *cron* job over a period of low database activity. When executed, it will prune data from the HBase database in accordance with some configurable criteria, such as the age of a particular data item for it to be considered as suitable for deletion.

The cleaner object will expose the `clean()` method without parameters. A hardcoded time limit within the program determines what is to be removed.

## 2.5 Event Monitor

This module acts as an interface between two separate components of the data pipeline: HBase and the PostgreSQL database which backs the system front-end.

While it is entirely appropriate for Hadoop to use HBase as a back-end data storage location for performance and scalability reasons, a NoSQL database is not necessarily the best format for integration with a front-end. For instance, it will be necessary for the front-end to display aggregated data, such as statistics concerning number of global packets or the number for a particular

router, perhaps grouped over some time frame. Retrieving this data from HBase directly and processing it in the front-end does not represent a good level of modularity. Such an implementation would place undue demands on hardware. It would not be particularly performant, nor would it be efficient for us to duplicate the well-tuned implementations of any of the freely available relational database management systems (RDBMS).

In order to interface between the two databases, it will be necessary to build a module which ferries data in an appropriate form between the two, extracting the necessary information and converting between the respective database schemas in the process. The Event Monitor, to be implemented as a Java service, will perform precisely this task. It will be notified of job completion by monitoring the Hadoop job queue to determine the point at which a job completes and fresh data appears in HBase.

## 2.6 PostgreSQL Storage Format

Our PostgreSQL database is responsible for storing the data to be represented by the GUI, as previously outlined. Two tables are expected to be required, and their schemas are provided below.

| Router | | |
|---|---|---|
| Column Name | Datatype | Description |
| routerIP | int | Unique identifier for the router that the data pertains to. |
| lastSeen | timestamp | The timestamp of the last known flow through this router. |
| flowsPH | int | Flows per hour. Calculated by moving average. |
| packetsPH | int | Packets per hour. Calculated by moving average. |
| bytesPH | int | Bytes per hour. Calculated by moving average. |

| Event | | |
|---|---|---|
| Column Name | Datatype | Description |
| eventID | int | Unique identifier for the event. |
| routerIP | int | Foreign Key (Router.routerIP). |
| ip | int | IP of attacker/victim within the BT network. |
| type | char(20) | Enumerated value for the type of attack. |
| status | char(20) | Enumerated value as to whether this event is active. |
| message | text | Any extra user-definable notes that are relevant to this event. |
| createTS | timestamp | The date and time the event were made. |

## 2.7 Python-based Web Server

### 2.7.1 Web data flow

The user interface provided by the system will be web-based. It will be powered by a Python backend which will poll from the PostgreSQL database at regular intervals, checking each time if new data has been processed by the Java backend. The database connection to PostgreSQL will be established using the PsycoPG Python library. Once new data is found, it can be converted into a Javascript Object Notation (JSON) format to be pushed to all web browsers that currently have the interface loaded. The use of a push will ensure that the system works to larger scales, since the amount of web traffic is not so dependent on the client's polling frequency. In addition to this, a connection may be made to HBase via the HappyBase Python library where graph information can also be included within the JSON data push.

A stream will be used to keep a connection active between the Python backend and the Javascript client. Redis will be used to handle the event stream data structure in Python and the system will use Server Side Events, a feature provided by HTML5, to push the JSON data to the clients. Although all browsers are expected to be standards compliant enough to support this, Microsoft Internet Explorer does not have a functional implementation, leading to the inclusion of a 'PolyFill' mimicing that functionality in incompatible browsers.

On the front-end, Javascript will be handling the connection to the event stream. On a new event, the data will be collected and parsed using the JSON Javascript class. The parsed data would then be sorted and sent to update the frontend. Any alerts, messages or extra information which is affected by the new event would also be updated at the same time. Most visualisation functionality will be provided by several Javascript libraries including Rickshaw JS and Flotcharts for graph visualisation and jQuery for information visualisation. The Javascript frontend will also be able to request further information, such as specific date ranges of data, in real time.

### 2.7.2 Flask and Web Serving

Flask is a Python library which allows rapid development of web based applications using its web framework. Flask starts a basic HTTP server on its own port and allows full control over what is sent when data is requested. The Flask web framework

will be run behind gunicorn to allow functionality such as server side event pushes. Nginx would then be used to run a reverse proxy for the HTTP server running on port 8000. Static files such as CSS and JS are also served through Flask.

The user interface will use other development tools that are known for their ability to speed up development and provide simple Javascript and CSS frameworks. These include Twitter Bootstrap, Rickshaw JS, and Flotcharts.

### 2.7.3 Justification to use Python's Flask web framework

The user interface is the most critical part of a system in regards to the ease with which a user can ascertain the information they have connected to the system to access. A Java Swing implementation was initially considered, but a number of issues were raised that led to the choice being dropped. It was considered that the interface would be difficult to create, aesthetically displeasing, inflexible and difficult to integrate into BT's existing systems, which currently use a web-based framework called Splunk. To consider the needs of the client, it was decided that the current system should be relatively 'plug-and-play' with regards to this knowledge, leading to an HTML-based system. Furthermore, the Java client model would require an installation of the client on every machine, whereas the web-based idea would simply require a URL being accessed within the browser.

A known method of connecting HBase to the web interface was via HappyBase and Python, and thus this was the method that was researched. Since the enterprise market is constantly migrating in the direction of consolidating data in a web portal, the general consensus amongst the developers was that the most important property the web interface could have would be to have long-term usefulness, something that was not envisaged under Swing.

# 3 Specific Requirements

## 3.1 Performance Requirements

The system works on a pipeline principle. As such, the slowest of all of the processes will dictate the total time that the data takes to be processed and displayed after input. The only hard requirement is that logs must not pile up in the input directory, which implies that the entire system must have a throughput greater than the filesize of logs generated on a per-day basis. Since this data has no requirement to be hard realtime, it is acceptable to be running at a slight backlog during peak hours if this entirely clears during off-peak, such as overnight.

## 3.2 System Attributes

### 3.2.1 Reliability & Availability

This system is not a network critical system. Its failure in any sense of the word must not cause BT's core network or any portion thereof to fail to route data for customers. With proper isolation from the core network, perhaps on a separate management network, this would not be the case.

However, there are many failure modes by which the system could fail. If business processes are adapted to incorporate this system as a critical part of the monitoring framework, it would be reasonable to assume the uptime expected of this system may not be dissimilar from the uptime expected of the core network itself. Furthermore, the increasing systemisation of monitoring routines can cause users to become dependent on such systems, which places further demands on the developers to ensure a high level of uptime, and to ensure the system produces reliable data.

The use of appropriate testing frameworks will be necessary to ensure the behaviour of each module conforms to its published documentation. For example, unit testing each Java module is an appropriate method, and will be the subject of the testing methodology elsewhere in this document. Such tests may not consider every failure scenario, nor is it possible to test how the system scales to many nodes without actually doing so. Finally, there is an implicit assumption that the third-party tools and systems employed on this project are free from bugs.

### 3.2.2 Handling of Private Data

The data that this system will be handling is sensitive, in that it pertains directly to customers of BT's network. It is entirely possible, from the packet logs, to not only identify one household as the source of the request, but to also identify the destination of the request. Using reverse DNS, this would even give a possible domain name associated with the packet, implying their current browsing habits. In order to consider data privacy laws and best practises, this information should not be displayed prominently in a human-readable manner, instead opting for graphs, charts and other aggregate displays to portray the state of the network to equivalent or better effect.

While it is possible to mask raw packet data from the users of the system, it will still be necessary to have access to IP addresses and other sensitive data about packet flow in the back-end infrastructure. This is inevitable for conducting data analysis, and can hardly be avoided. However, this does raises two issues:

The data will move between many systems as it flows through the data pipeline, including different databases and different services, each of which will modify the data in various ways. Each system is likely to sit on one or more servers, and in the case of Hadoop and HBase, will reside across many servers in their respective cluster. This places demands on the physical security of the infrastructure on which this system will operate, such that the physical machines should be located in a secure datacentre environment. In addition, it would be preferable for the data to be exchanged between systems in a secured environment, preferably isolated entirely from the Internet. If data must transit untrusted or insecure networks, it will be necessary to enforce some degree of encryption on such connections, although that is beyond the scope of this project. Similarly, if the terminals used by the users are not known to be secure, some internal access control system may need to be implemented on the front-end to prevent unauthorised access. Through efficient code development, it is hoped this would not be too much work to integrate with existing systems.

### 3.2.3 Maintainability

It is expected, at completion of the project, for the code to be provided in a suitable format for integration with existing systems and for future development work by the client's engineers to be possible.

To facilitate this, the behaviour of all code will be documented in-line. Additional notes as to interfaces and design decisions will also be provided on completion where these are considered appropriate, or where it may not be possible to derive the intention behind particular design decisions from the code itself. The well-known *Javadoc* method of documentation will be used in the methods of Java code for, at the very least, any method exposed as a public interface.

By virtue of the effective use of modularity to subdivide the project into independent components, it is expected that each module will be clearly defined in the codebase. In addition, documentation such as this is produced partly for the purpose of future use within BT, as high-level design decisions and the data flow through each module are presented for the benefit of future

developers. The intention is that such coding standards are simple to implement on-the-fly as code is written, so they may be continued in future iterations of project development with little effort.

#### 3.2.4 Portability

There are a considerable number of modules which comprise this project, which suggests its final integration will be complex. This is to be expected of any system whose data processing will be distributed over many nodes and in which there are specific requirements to present data in a particular manner to the end-user. Many of the modules will be tied to particular machines or infrastructure which should already be in place, such as a Hadoop cluster and the presence of a reporting framework for the routers. Nevertheless, the modules have complex interactions with each other, which somewhat restricts the possibility of moving particular services or daemons to other hosts on a whim.

In addition, there are specific requirements on the Hadoop and HBase frameworks which the project will be closely coupled to. The virtual machine environment provided by Cloudera for Hadoop processing will be used. This is a common Hadoop implementation across many enterprises, and complements BT's usage of the same provider. The code could, in principle, work with any vanilla Hadoop installation, but certain aspects may be restricted in functionality or broken if the versions are dissimilar. Cloudera CDH4 will be the specific release used for this project.

The version of Java used will be Java 6, since Cloudera control the update servers for the distribution, and that is the version they provide for download. However, there is no reason for a migration to a newer version of a JVM - such as a Java 7 JVM - to be impractical when the Cloudera image is updated.

There is no reason the finished product could not be utilised on a network other than BT's. However, many decisions have gone into the design of the algorithmic analysis and presentation of data which are specific to BT's use cases, closely coupling the final product to their business processes. Porting the product to monitor another provider's core network, such as in the case where BT wished to license it as a product, would involve a non-trivial amount of work. Design decisions must be generalised and adapted to the specific hardware in the network, should it not provide the same input file format, and to the types of data that the provider expected to be able to retrieve, since each MapReduce job is specifically intended to answer a certain question about the state of the overall network.

### 3.3 Testing Methodology and Acceptance Criteria

#TODO

# 4 Group Management Strategy

The group has come to a consensus that none of its members will assume the role of Project Leader. It was felt that decisions as to the design or management of the project would impact all team members, and thus they had every right to express their views upon it. As such, the delegation of the work has largely been such that each module is assigned to the member that displays the greatest aptitude for the task, by agreement with all members of the group. For complex modules or those which are time-consuming to implement, at least two members have been assigned to prevent overloading any particular member of the group.

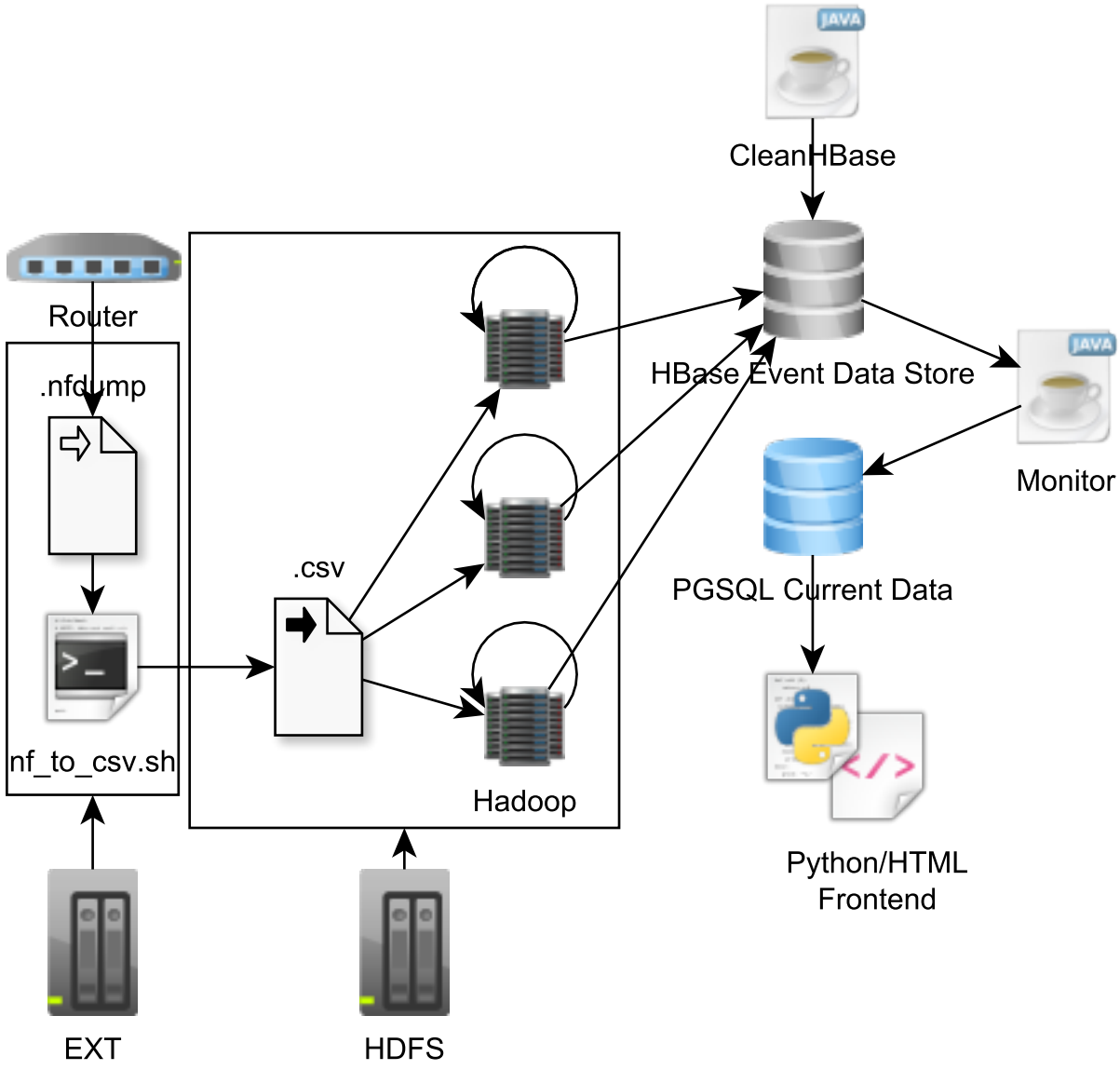
The member or members assigned to the implementation of each module are responsible for the design of its public interface, the flow of data within the module, and for providing routine feedback to the group as to their progress on implementation by supplying appropriate documentation.

Each module has been assigned to group members as follows:

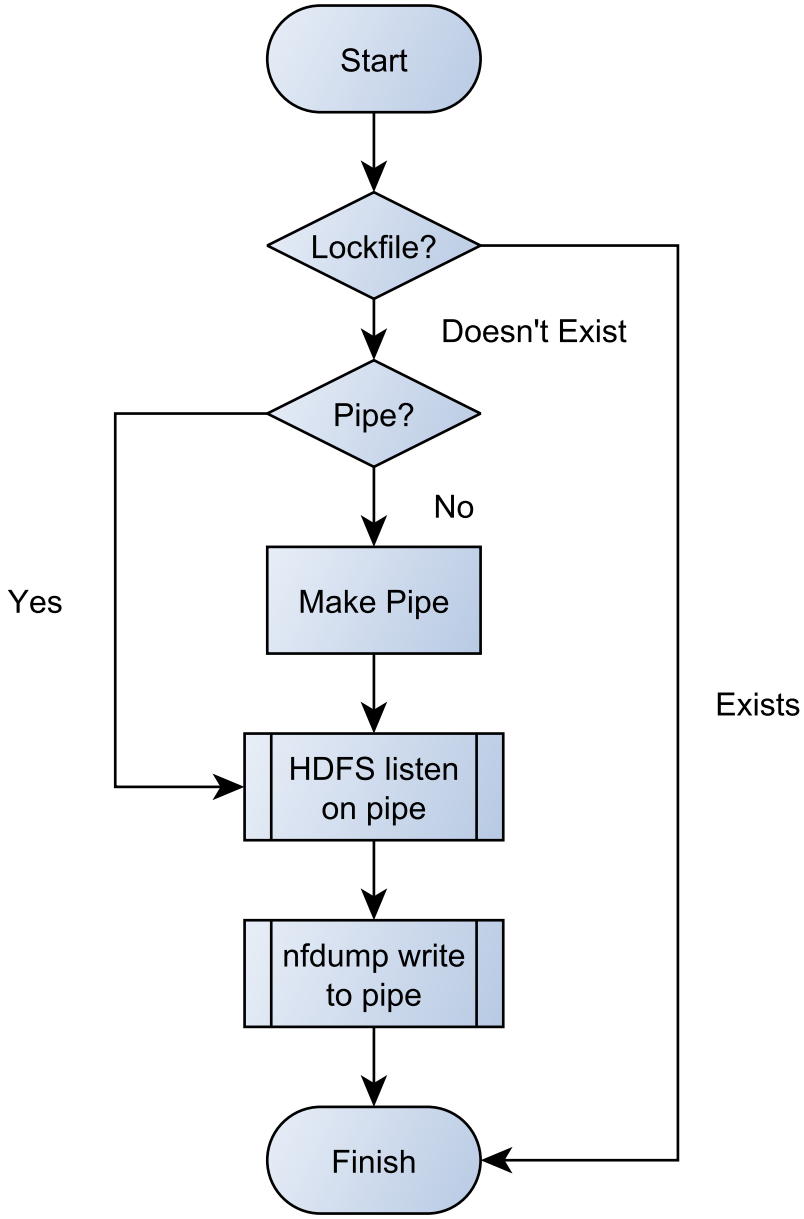
- File Importer: Simon Hollingshead and Matthew Huxtable.
- MapReduce Jobs: Jan Polášek, who will implement the algorithmic analysis of the data.
- Hadoop Infrastructure and Control framework: Aaron Kirkbride and Ernest Zeidman.
- HBase DB Storage: Aaron and Ernest.
- HBase Cleanup Utility: Alex Marshall.
- Event Monitor: Simon and Matthew.
- PostgreSQL Database Administration: Aaron and Ernest.
- Python webserver: Aaron.

In addition to the implementation of the modules, there is also a significant amount of formal documentation to be prepared as the project progresses. The documentation for public interfaces and implementation-specific details will be provided by each team member. However, there is also a general requirement for documentation of the project as a whole and for collation of the information provided by each team. It was felt that having each member of the team write significant amounts of documentation would detract from the time available to them to implement their modules. Instead, this responsibility has been assigned to a specific team as if it were a module of sorts, and will fall to Simon and Matthew.

# A Overall System Overview

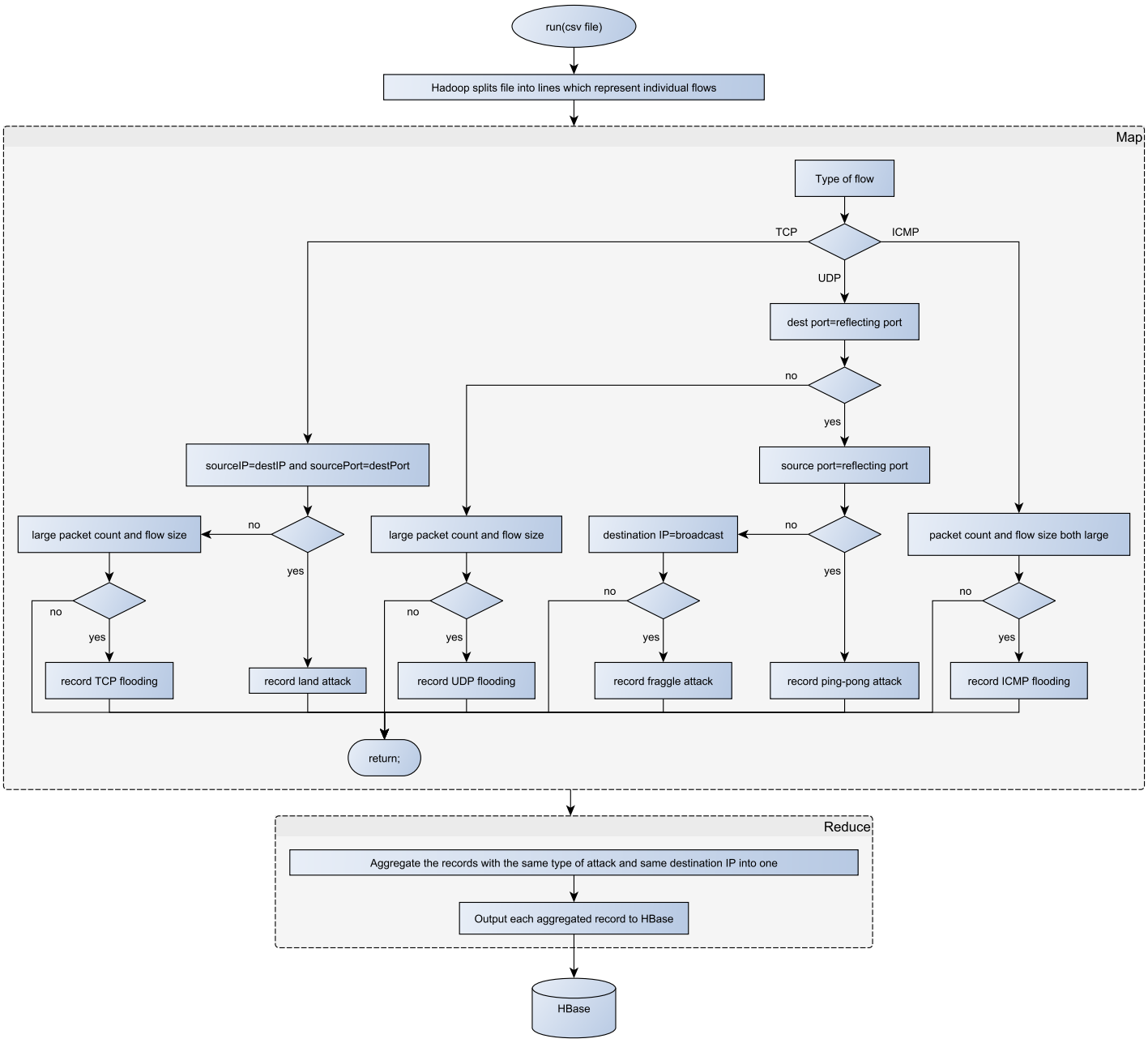


## B Importer Tool Flowchart

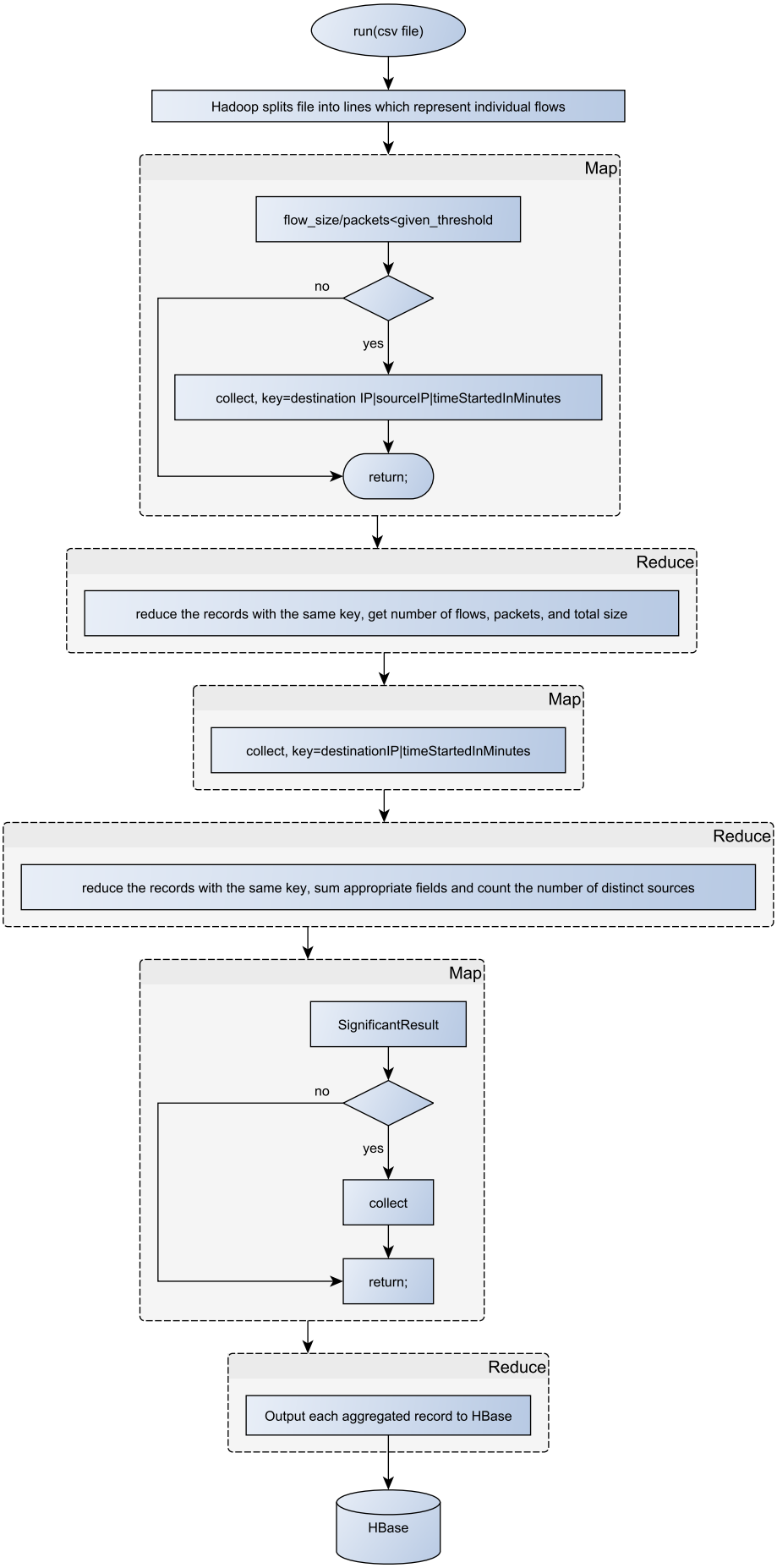


# C MapReduce

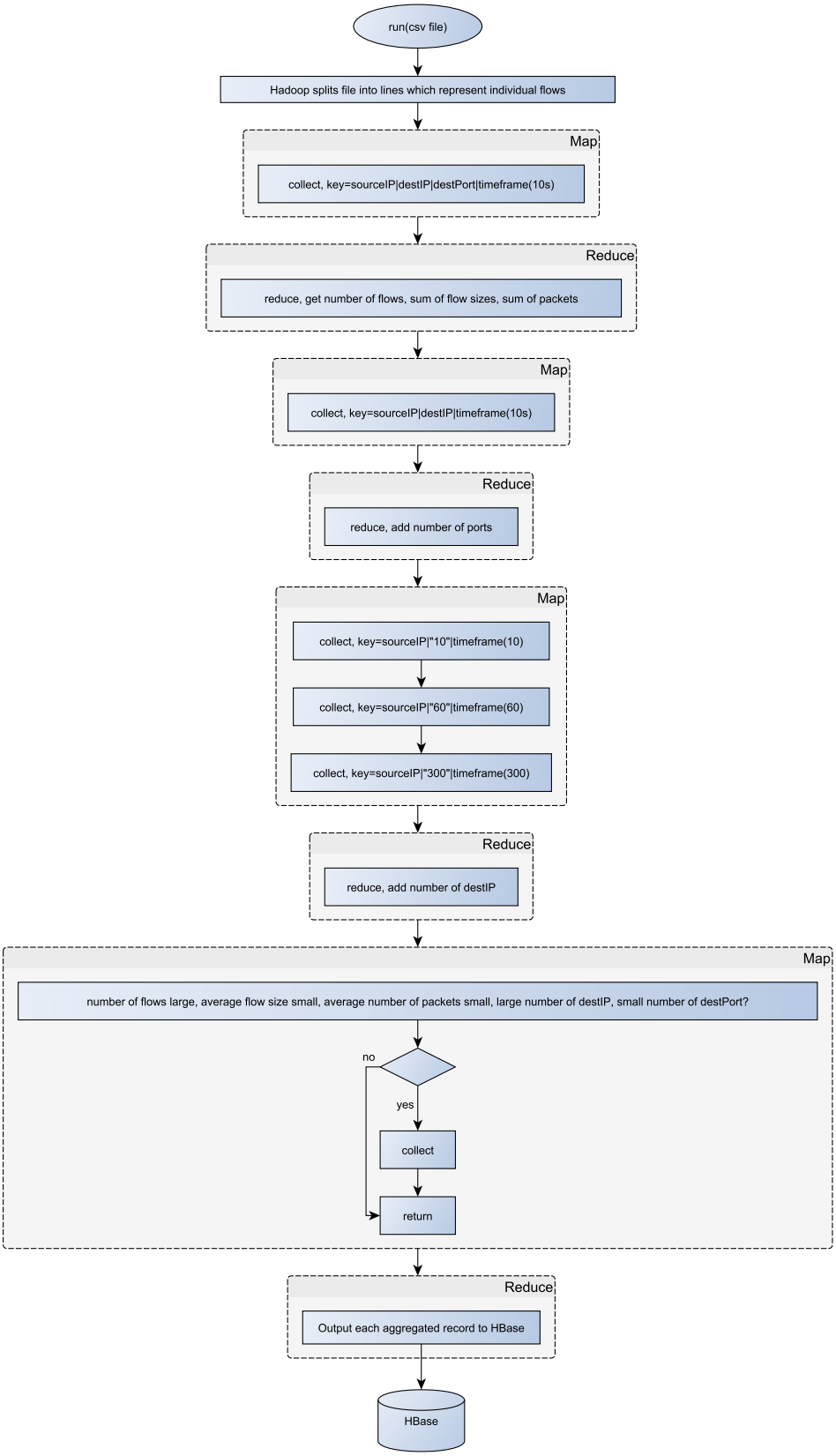
## C.1 Single-flow Threats



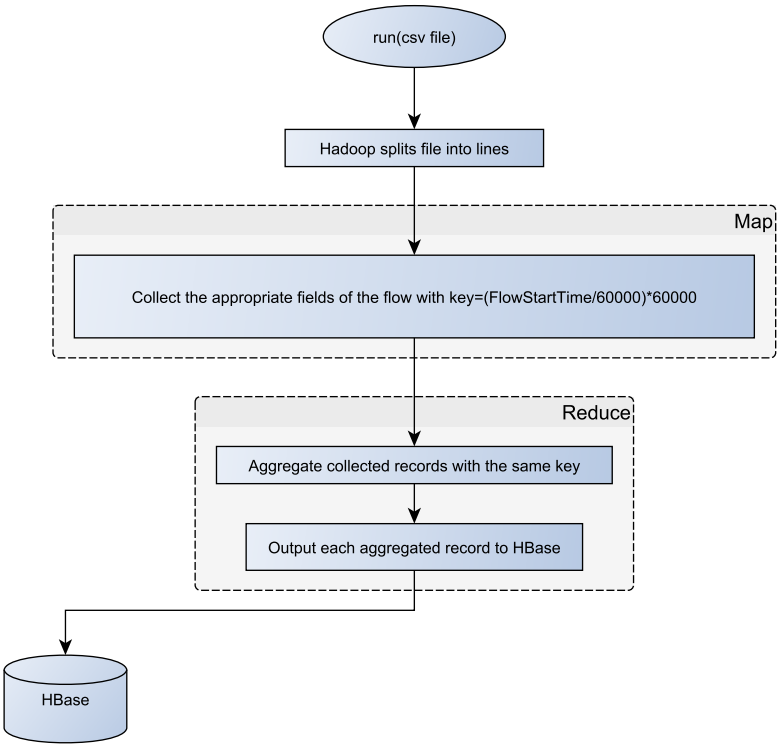
## C.2 DoS Attacks



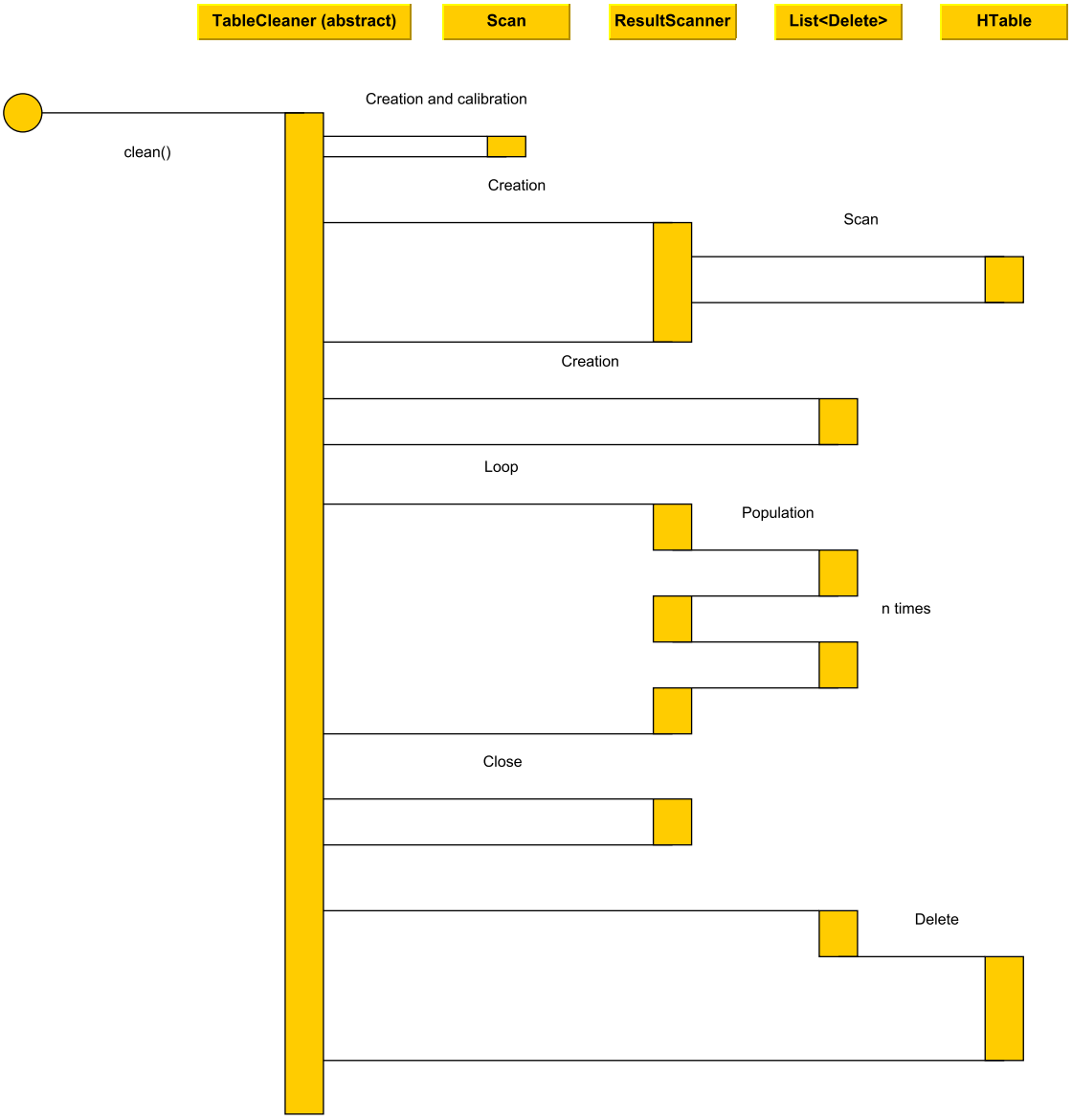
## C.3 Port Scanning



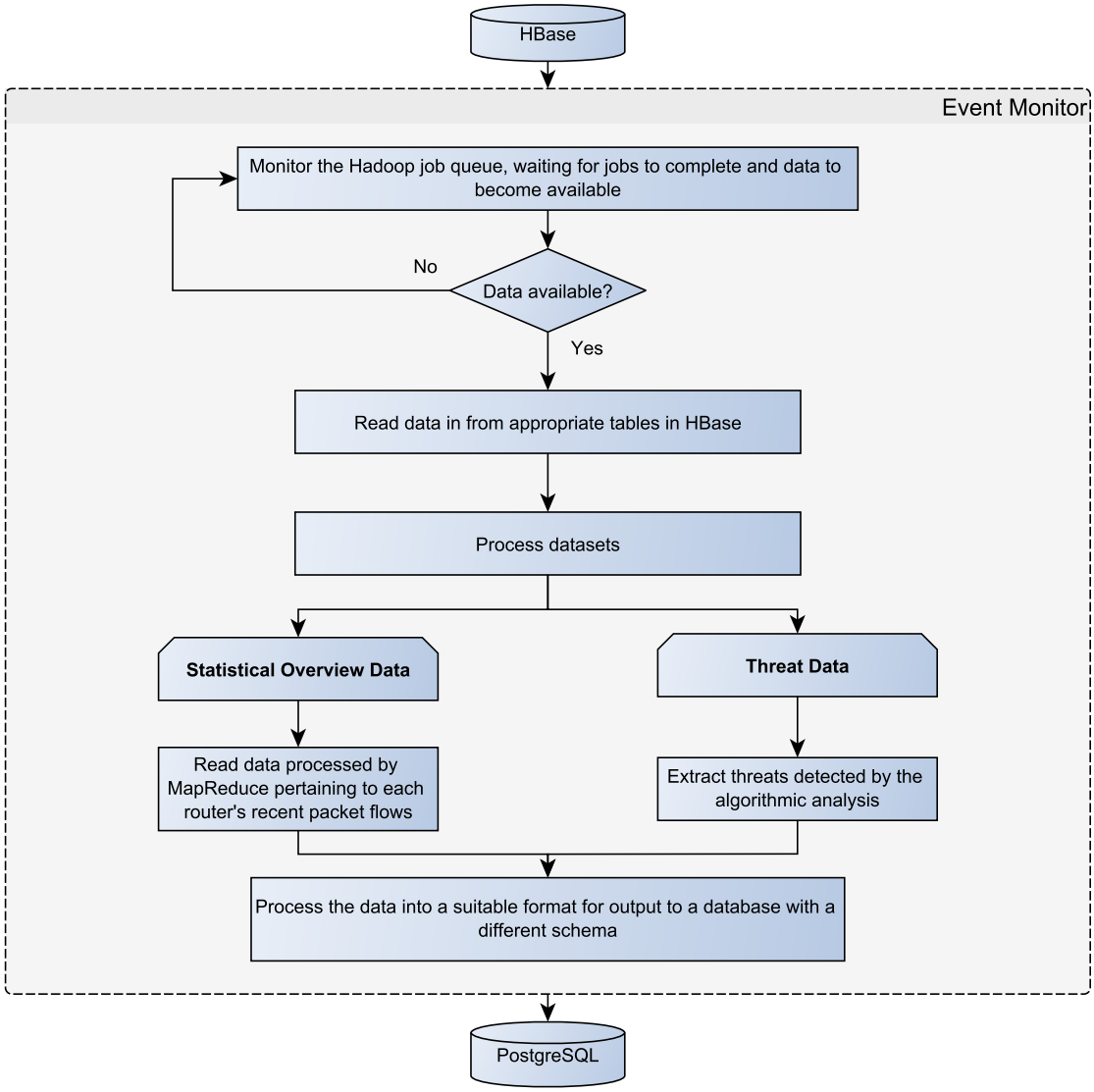
## C.4 Statistics



# D HBase Cleanup Flowchart

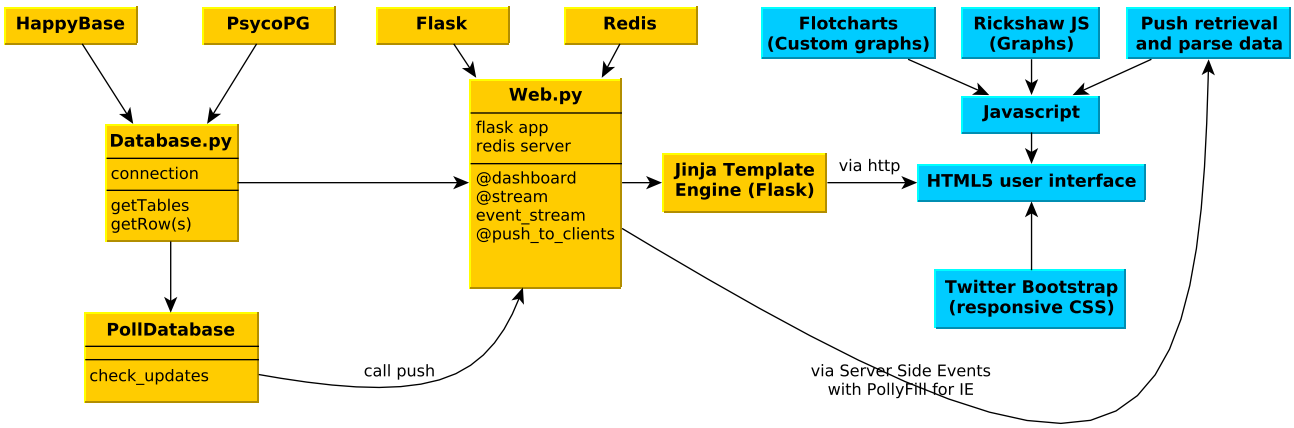


# E Event Monitor Flowchart



# F Web Server

## F.1 Class Diagram



### F.2 Data Flow Diagram