Data Visualization for Discovery of Digital Evidence in Email

André Nordbø

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2014

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

Digital recordings of our activities are constantly being stored and processed in information systems, and are increasingly getting more valuable for investigating illegal activities. A problem we are facing today is the amount of manual labor required in order to look through this ever-growing amount of data. This process is time consuming and error prone. Tools and methods for speeding up the process are needed, and they must be capable of aiding both identification and correlation of the evidence. An indirect benefit is reduction in human error by freeing up human cognitive processing.

The topic of this thesis is email investigation and how visualization techniques can support it. Data visualizations can help humans spot trends, correlations and anomalies in the data. Anomalies are of particular interest, based on the assumption that illegal behavior can be detected indirectly based on outlier characteristics of meta data. Two sources of email data are used. The primary source is the Enron dataset consisting of approximately 150 corporate accounts. The second data source is a private Gmail account synchronized via Microsoft Outlook.

The research methodology comprises design, implementation and testing of a modular framework for email investigation. It has three main parts: Standardization of email format, extraction of meta data and a web based interface. Preliminary testing comprises demonstration of designed techniques using the Enron dataset, verification of email parsing using the Gmail account and lastly verification of visualization results using a commercial tool called Tableau. Standardizing on a common format before meta data extraction eases the process of adding support for new or changing email storage formats.

Important steps in meta data extraction are removal of duplicates, determination of owners' email address(es) and determination of direction. Deleted messages that still remain in other user accounts can be recovered. Email accounts that have not been collected can partly be rebuilt based on the same principle. Extracted meta data have been imported to the commercial tool Tableau which proves to be an efficient environment for prototyping visualizations.

Important findings for interaction and visualization are the benefits of interlinking visualizations with the underlying data. Interlinking both in time and across email accounts. The sending direction of a message is important when visualizing the *time of day* or number of *messages per day* since sent messages correlate stronger with user actions than received messages. Moving analysis of email from a desktop application to a web portal opens up for new ways to collaborate. Investigators can see what messages has been read by others and important messages can be added to a shared case timeline.

A working prototype with support of Microsoft Outlook *.pst files has been prepared, and can be used for further research.

Preface and Acknowledgements

I have always appreciated the information transfer capacity of data visualizations done correctly. How it can ease understanding of complex data and concepts. This is the main motivation for the choice of topic and I hope this work will inspire and motivate increased usage of visualization techniques for aiding forensic investigations.

First I would like to thank my supervisor Katrin Franke for good advice and inspiring discussions during these 5 months of working on my thesis. I also thank my co-supervisor Slobodan Petrović for quick answers of my questions and his references to interesting reading material.

Many thanks to the people at Økokrim for hosting a midterm meeting giving us us the opportunity to present and get valuable feedback on our progress. Our discussions and getting to know their workflow has been very helpful in prioritizing the focus of the thesis.

Being part of a community of peer students has been invaluable. I especially thank Pieter Ruthven, John Erik Rekdal, Dmytro Piatkivskyi and Espen Didriksen for motivational support, interesting discussions and peer review efforts.

This work builds on many existing open library packages. *LibPST* by Dave Smith and Carl Byington for parsing of PST files. *OleFileIO_PL* by Philippe Lagadec and Fredrik Lundh for parsing of MSG files. For interaction and visualization: *jQuery*, *jQuery Sparklines* by Gareth Watts, *d3* by Mike Bostock, *nvd3* and *Timeline* from the CHAP Links Library by Jos de Jong. A special thanks to Renato Pereira, the author behind the *liac-arff* Python library for reading and writing of ARFF files. He quickly updated and fixed issues discovered in the code base during my initial experiments with data formats.

Contents

Abstract . . . . . **ii**
Preface and Acknowledgements . . . . . **iii**
Contents . . . . . **iv**
List of Figures . . . . . **vii**
List of Tables . . . . . **ix**
1 Introduction **1**
1.1 Topic description 1
1.2 Keywords 2
1.3 Problem Description 2
1.4 Justification, Motivation and Benefits 4
1.5 Scope 4
1.6 Research Questions 5
1.7 Methodology 5
1.8 Claimed contributions 6
1.9 Thesis Outline 7
2 Background **8**
2.1 Forensic principles 8
2.2 Types of evidence 10
2.3 Visualization theory 12
2.4 Anomaly detection methods 16
2.5 Related works 21
2.5.1 Differential analysis 21
2.5.2 Stream of events 29
2.5.3 Social network data 33
2.6 Testing methodologies 40
2.7 Enron 41
3 Method design & implementation **42**
3.1 The email format and guiding questions 42
3.2 Method considerations 45
3.2.1 Message extraction module 45
3.2.2 Parser module (feature extraction) 47
3.2.3 Output format 48
3.2.4 Visualization and interaction 49
3.2.5 Filtering options 52
3.3 Implementation 53
3.3.1 Email extraction: Converting PST files 55

3.3.2 Meta data extraction: Parsing EML files . . . 56
3.3.3 Visualizing using Django and JavaScript . . . 58
4 Experiments, results and discussions . . . 61
4.1 Experiment 1: Enron . . . 62
4.1.1 Test setup . . . 62
4.1.2 Results: Meta data extraction . . . 62
4.1.3 Results: Visualization and interaction . . . 66
4.1.4 Discussions . . . 73
4.2 Experiment 2: Outlook PST file . . . 75
4.2.1 Test setup . . . 75
4.2.2 Results . . . 77
4.2.3 Discussions . . . 79
4.3 Experiment 3: Tableau software comparison . . . 80
4.3.1 Test setup . . . 80
4.3.2 Results . . . 82
4.3.3 Discussions . . . 84
4.4 Anomaly detection summary . . . 85
5 Conclusions and further work . . . 86
5.1 Research questions answered . . . 86
5.2 Theoretical implications . . . 89
5.3 Practical considerations . . . 89
5.4 Further work . . . 90
5.4.1 Testing on expert investigators . . . 90
5.4.2 Extending the framework . . . 90
5.4.3 Optimizing code . . . 91
Bibliography . . . 92
A Missing headers using libpst . . . 100
B Automatic determination of email address . . . 101
C Output of the dataset overview . . . 104
D Implementation details . . . 108
D.1 Parsing PST and MSG files . . . 108
D.2 Meta data extraction: Parsing EML files . . . 108
D.3 The django module . . . 112
E Source code . . . 118
E.1 runall.py . . . 118
E.2 parse_pst.py . . . 118
E.3 parse_msg.py . . . 119
E.4 metadata.py . . . 121
E.5 direction.py . . . 127
E.6 undelete.py . . . 129
E.7 timestatistics.py . . . 130
E.8 recover.py . . . 132

E.9 tableau.py 134
E.10 server/settings.py 135
E.11 server/urls.py 135
E.12 server/views.py 136
E.13 server/models.py 138
E.14 server/overview.py 138
E.15 server/usertimeline.py 140
E.16 server/usertod.py 142
E.17 server/usercom.py 142
E.18 server/messages.py 143
E.19 server/templates/index.html 144
E.20 server/templates/overview.html 144
E.21 server/templates/usertimeline.html 148
E.22 server/templates/usertod.html 150
E.23 server/templates/usercom.html 152
E.24 server/templates/messages.html 155
E.25 server/templates/casetimeline.html 163

List of Figures

1 The forensic process by NIST SP800-86 . . . 9
2 Visual data exploration framework . . . 15
3 Illustration of sample and attribute . . . 17
4 Cyber Forensic TimeLab . . . 21
5 Perspective Wall . . . 22
6 Change-link: Windows Shadow Copy visualization . . . 23
7 Malware cross correlation . . . 25
8 Self organizing maps for file system meta data . . . 27
9 Webscavator: Visualizing web history . . . 28
10 Protocol classification based on inter-packet timings . . . 29
11 TCPflow capture visualization . . . 30
12 Splitting a URL into substrings . . . 31
13 Graph visualization of web access requests . . . 31
14 Visualized Bayesian classifier . . . 32
15 Visualized Bayesian classifier with sliding window . . . 32
16 Ontea: Enhancing email with contextual information . . . 33
17 Enronic: Visualization of text classification . . . 35
18 Threaded conversation graphs . . . 36
19 Email Extraction Tool . . . 37
20 i2 analyst notebook . . . 38
21 Pixel based daily account activity . . . 39
22 Plaintext email example with 7-bit ASCII . . . 42
23 Plaintext email example with quoted-printable encoding . . . 42
24 Email message with special characters . . . 43
25 Overview of email framework . . . 45
26 Proactive email extraction . . . 45
27 Reactive email extraction . . . 46
28 Adding back deleted messages . . . 47
29 Design for the Dataset overview . . . 49
30 Design for user activity level . . . 50
31 Design of a time of day plot . . . 50
32 Design of communication breakdown . . . 51
33 Design of message content view . . . 52
34 Implementation overview . . . 54
35 Determination of email address . . . 56
36 Message recovery candidates . . . 57

37 Meta data summary . . . 57
38 Dataset overview implementation details . . . 58
39 Message content view: Field details . . . 59
40 Example: Case timeline . . . 60
41 Distribution of addresses in the From header . . . 64
42 Dataset overview: huge gaps . . . 66
43 Dataset overview: Zoom in . . . 67
44 Example: Weekly timeline . . . 67
45 Example: Message content view . . . 68
46 Time of day view: Mark Taylor . . . 69
47 Time of day: Same pattern . . . 69
48 Communication breakdown: Mark Taylor . . . 70
49 Detailed message view . . . 70
50 Weekly timeline: High weekend activity . . . 71
51 Example of message filtering . . . 71
52 Example: breakdown for tana jones . . . 72
53 Outlook: Download preferences . . . 75
54 Outlook: Location of PST file . . . 75
55 Outlook: Original headers . . . 76
56 Gmail account: Dataset overview . . . 77
57 Gmail account: Day of week and time of day . . . 78
58 Gmail account: HTML preview and attachments . . . 78
59 Gmail account: User communication breakdown . . . 79
60 Tableau: import of email . . . 81
61 Tableau: all account activity . . . 82
62 Tableau: Recovered messages in time . . . 83
63 Tableau: Weekly account overview with stacked weekdays . . . 83
64 Tableau: Time of day . . . 84
65 File description of the Django project . . . 112
66 Django urls.py . . . 113
67 Weekly timeline controls . . . 115
68 Time of day data . . . 115
69 Time of day example . . . 116
70 Python Pandas GroupBy . . . 116
71 Communication breakdown data format . . . 117
72 Communication breakdown visual modes . . . 117

List of Tables

1 Comparison of message and meta data size 62
2 Users with email address mismatch 63
3 Duplicate removal and undelete 65
4 Speed of code on Enron dataset 65
5 Comparison of message and meta data size for PST file 77
6 Duplicate removal and undelete for PST file 77
7 Speed of code for PST file 77
8 Enumeration of extracted meta data, part 1 109
9 Enumeration of extracted meta data, part 2 110
10 Enumeration of extracted meta data, part 3 111
11 Enumeration of extracted meta data, part 4 111
12 Data collected by the Django overview function 113

1 Introduction

1.1 Topic description

In the last few decades we have witnessed an enormous increase in the usage of digital information systems. These systems consist of interconnected computing devices, typically connected via the Internet. Rapid evolution in terms of digital storage and processing power has enabled archiving and processing of increasing data volumes of our activities. Both in business and in private life.

This high technology has made possible huge gains in productivity. The speed of communication together with automation of previously manual tasks are key factors. Recent trends in interconnecting everything is known as the "Internet of things". It focuses on how to interconnect your car, refrigerator and perhaps even your shoes using digital technology. What we end up with is an enormous network of sensors collecting all sorts of data continuously.

Using high technology for efficiency gains comes at the price of new vulnerabilities and a reliance on these information systems. Most of us have very limited insight into how the technology works, yet we rely on availability, confidentiality and integrity of the data trusted to them. Every human society has people taking advantage of opportunities in unaccepted ways. They range from mere criminals to state sponsored actors.

The field of digital forensics focus on collecting and analyzing residual digital traces with the goal of answering questions related to misuse. Has security been breached? Has a law been violated? Was something stolen? Who was behind it? Potential evidence is secured and analyzed.

The focus of this thesis is the usage of computational methods [1] for aiding forensic investigators in discovery and analysis of digital evidence. Data visualization techniques in particular, and for correlation of data across multiple sources [2]. Data visualization is a technique for data transformation for easing the human cognitive perception of it. Often using visual graphical representations. Humans are especially adapt to visual information and are very good at spotting anomalies and trends.

Anomaly detection can be explained as identifying what is not normal. Synonyms are outliers, exceptions or strangers. An outlying observation in a dataset can be thought of as an erroneous reading and thus a candidate for removal, but in forensics such an anomaly would be the focus of investigation. The assumption would be that deviations from the norm have a higher probability of being candidates for suspicious or even illegal activity. Examples could be sudden dramatic increases in systems log size indicating an intrusion attempt, or people suddenly working a lot outside of normal working hours perhaps indicating something critical has happened. Both of these situations could also have explanations such as a product release causing more visitors or an upcoming deadline causing people to work extra hard to finish on time. It is still reasonable to assume pointing to potential needles in the haystack is of value.

1.2 Keywords

Digital Forensics
Email analysis
Computational Forensics
Data Visualization
Evidence Correlation
Anomaly detection

1.3 Problem Description

The forensic science argues on matters related to criminal or civil law based on evidence. The word forensic comes from Latin and means "of or before the forum." [3] emphasizing the end goal is decision making based on reconstruction of past events [4]. It is of utmost importance that evidence presented is trustworthy, understandable and based on thorough examination of the available data. The consequence could otherwise be sending an innocent person to jail or releasing a guilty person because of lack of evidence.

Courts rely on expert witnesses to testify on the interpretation of digital evidence. Human experts will look for evidence based on intuition, their "gut feeling". Intuition is different from person to person and humans make mistakes. Especially under pressure. In order to achieve equality under the law, procedures need to be standardized. This is a major motivation for the Daubert rulings in the United States.

The Daubert standard regulates the admissibility of expert testimony when dealing with scientific and technical evidence. Admissibility is the regulation of what evidence can be accepted in a court of law. Judges shall decide whether methods used in testimonies are scientifically valid and relevant for the particular case [5]. Important guidelines are whether the method is peer reviewed via publications and is generally accepted in the scientific community. Methods should be testable, preferably with estimates of error rates. The Daubert standard extends the Frye standard which only required acceptance in the community. The authors of [6] arguments scientific knowledge evolves over time. Publication and peer review does not guarantee the conclusion as it mainly focuses on novelty and methodology. The responsibility of determining the level of conservatism to new methods is left to the judges, often lacking special training in scientific methodology [6]. This highlights how important it is to be able to explain why a method works and be able to back it up with empirical data.

The increased diversity of digital devices, operating systems and applications keeps complicating the task of preserving and examining digital data [7][8]. The relevant digital evidence is typically hidden or attempted destroyed. It is mixed with mostly benign data [1]. Anti forensics tools such as encryption and secure erasing deny forensic examiners access to some of the data. Storage off-site can deny local access but at the same time opens up back doors to the data

through the service providers hosting it [7]. The need to cross correlate across multiple sources has increased drastically because of this, and will continue to do so as an effect of the "Internet of things" trend. It is important to realize that although anti-forensics techniques exist, often times they are not actually used. The problem is often too much available data [9][10].

The scalability problems we face [11][12] has two aspects: One is speeding up the collecting and processing of the data and the other is how to speed up the cognitive analysis of the processed result. Contemporary tools available are written mainly to run on workstations and focus on decoding of disk copies. This is done by reading the file systems on them, locate deleted and hidden files, index it and make it available in human readable form so that the forensic investigator can read and search it. This process limits collaboration and correlation across data sources.

Thorough investigations are time consuming and the time required increases in proportion to the amount of available data. One way to solve this problem is to put more human resources at it, but an even better way is using computational power to help examine the data. Data visualization is a promising solution to the problem. Tools for the next generation forensic tools also need support for automatic analysis techniques to guide the forensic examiner to the relevant data across data sources using algorithms that adopt to feedback from the investigator [13].

The problem statement can thus be summarized to: Current tools for analysis of digital data scale poorly with the steadily increasing amount of it and lack support for correlation and collaboration. The proposed solution is to study how data visualization can aid analysis.

1.4 Justification, Motivation and Benefits

Visualization techniques where humans are supported by computers is a promising approach. Most abnormal actions are not illegal, but the probability of an illegal action being abnormal is assumed to be high. Presenting the forensic analyst with visualizations of the data and easy access to correlate across data sources can help reduce backlogs of work and reduce human error. This as a consequence of reducing tedious manual labor.

Many tools and methods have been proposed in the literature, but the adoption rate by forensic investigators is claimed low. The reason can be because they are not easily available, too difficult to use or lacks maintenance and support [7]. In any case it seems like there is unrealized potential that would benefit increased awareness by showing how visualization techniques can speed up analysis.

Faster discovery of important evidence increase the probability of getting to the correct conclusion. Society will benefit by being able to deal with the increased usage and exploitation of high technology for criminal and malicious gain.

1.5 Scope

Email was the chosen data source. It is wide spread with approximately 3 billion accounts as of 2011. The number is expected to be 4 million in 2015 and approximately 25% of the accounts are corporate [14][15]. An email leaves a written trace as compared to a phone call and is consequently heavily used in many digital forensic investigations [16]. This thesis focuses on structured meta data extracted from email, such as when a message was sent and the participants involved in the conversation.

Corporate employees typically use email both for internal and external communication. Investigating individual email accounts one by one is not efficient. What is needed to to explore ways to ease cross correlation of multiple accounts at the same time. A primary goal is to explore how visualization techniques and web technology can speed up discovery of digital evidence. Commercial tools for email analysis do exist. They are often very expensive and closed source. Extending these tools is not an attractive option for further research since implementation details are hidden. The secondary goal is therefore to design and implement an open platform that can support further research into email forensics.

The Enron dataset was chosen because it has multiple tightly interconnected email accounts. It contains authentic email of a large company accused and convicted of economic crime.

1.6 Research Questions

- Q1: What visualization techniques exists for aiding digital investigations in published literature?
- Q2: What (meta) data and preprocessing are being used to drive these visualization techniques?
- Q3: What questions would a forensic expert want to answer from collected email?
- Q4: What functionality is needed of a framework for supporting research in email?
- Q5: Can the investigative questions aid design of interesting visualizations?
- Q6: In what ways can an implementation be benchmarked?
- Q7: What are the benefits and limitations of anomaly detection methods in terms of aiding email forensics?

1.7 Methodology

Scientific methods can be categorized into qualitative and quantitative methods. Typical methods seen in this field of study are development of prototypes and then testing them with a combination of quantitative and qualitative steps. Qualitative steps can be simplified scenarios or collecting subjective user experience in order to show benefits of the method. Quantitative methods can be measuring time and quality of performing selected tests to estimate user performance when comparing different methods or tools.

- Q1: What visualization techniques exists for aiding digital investigations in published literature?
- Q2: What (meta) data and preprocessing are being used to drive these visualization techniques?

A fundamental principle of scientific methodology is to build on existing work. The 1st and 2nd questions are targeted at gaining an overview of current literature and tools. First a broad literature study is conducted. It will look at how visualizations have been used in digital forensics in general. It is motivated as a means for gaining inspiration for ideas without limiting to email. The literature review then narrows down on email specific publications. Research into creation and preprocessed of the Enron dataset is also required.

- Q3: What questions would a forensic expert want to answer from collected email?

When an investigator digs into the available data he or she has questions in need of answers. The strategies for answering the 3rd question are ideas from the literature study, discussions with supervisor and gained experience from the authors studies so far. It will be based on what the email format contains of information.

- Q4: What functionality is needed of a framework for supporting research in email?

- Q5: Can the investigative questions aid design of interesting visualizations?

Abstraction and modularization are two very powerful techniques to deal with complex problems. Modularization is the art of creating abstract separated functionality blocks with defined input and output. A perfect example of modularization is the layered model of the Internet where information bearers can be replaced and additional application layer protocols added without rewriting the entire information chain.

The existing framework by NIST [17] of data collection, examination and analysis makes it clear that these tasks can be separated. Collection is typically performed using dedicated tools because of the variety of physical storage medium and the need for extracting data from live systems. Examination and analysis on the other hand are often combined in the commercial forensic tools and this is where separation is needed [7]. The examination step should take as input raw data. It could be disk images, memory dumps, network captures or other data structures. The output could be extraction of all images, email messages or written user documents depending on the case specific scenario and should be in a standardized format. The analysis step can now be implemented with this defined input and it makes it easier to experiment and test alternative analysis methods because efforts to integrate previous steps are now independent.

A design and prototype implementation of a framework for email investigation is developed in order to answer question 4 and 5. An iterative experimental process with focus on understanding the data: Look at the data, find interesting patterns and implement ways to show them. Python will be used for preprocessing and web technology for visualizations and user interface.

- Q6: In what ways can an implementation be benchmarked?
- Q7: What are the benefits and limitations of anomaly detection methods in terms of aiding email forensics?

Question 6 will be answered by running quantitative and qualitative experiments on the developed prototype using the Enron dataset. Verification of the examination step is then performed using an up to date dataset. A general purpose visualization tool is then used for showing the benefit of having examination and analysis separated by a standardized format. Question 7 will be answered based on what is learned during this design and development phase.

1.8 Claimed contributions

- A modular design for extracting meta data from multiple email accounts, and visualizing it using web technology.
- A prototype implementation of the design with support of Microsoft Outlook *.pst files and a working environment for navigating a large campus of accounts based on extracted meta data.
- Testing of the implementation using the Enron dataset and a Google Gmail account. Testing consists of showing how meta data can be visualized in order to answer investigative questions, and also demonstrate how efficiently novel visualizations can be designed based on extracted meta data using the general purpose visualization package Tableau.

1.9 Thesis Outline

The following chapters are:

- *Background*: A general introduction to the field of digital forensics, data visualization and anomaly detection theory. Following are examples of related visualization and machine learning techniques, first for the field in general, narrowing down to email forensics. The chapter ends with examples of testing methodology and description of the Enron company and dataset.
- *Method design and implementation*: Visualization theory is combined with investigative questions in order to design and implement a modular platform for email investigation using web technology.
- *Experiments, results and discussions*: Three experiments document how the implementation performs on the preprocessed Enron dataset and a more realistic Gmail account belonging to the author. The implementation is compared with the general purpose visualization tool Tableau Desktop before summarizing with a discussion on automatic anomaly detection.
- *Conclusions and further work*: Answers to the initial research questions and directions for further research.
- Bibliography, appendices and source code.

2 Background

This chapter presents an introduction to the field of digital forensics followed by theory on data visualization and anomaly detection. This introduction is followed by examples of related work. The initial examples are general to the field of digital forensics while the later ones narrow the focus to email investigation. These examples answer questions related to visualization techniques and testing methodologies. What data is used for visualizing, how is the data preprocessed and how can the resulting visualization aid investigators.

Testing methodologies has been separated out into a dedicated section. This chapter ends with a brief introduction to the Enron dataset used for experiments.

2.1 Forensic principles

Digital forensics (formerly computer forensics) uses scientific methodology to collect and analyze digital data independent of media [4]. The goal is to determine probable underlying causes given the traces left behind. It uses a hypothesis driven approach where digital evidence are either in support of or in dispute of a predefined hypothesis. An example of a hypothesis can be that a company has performed a financial crime. Sources of relevant digital data will typically be secured such as laptops, smart phones and corporate servers. Evidence could then turn out to be specific email messages and transaction records in payment systems where the content is in opposition of accepted laws.

The forensics science has traditionally been a reactive post-mortem activity meaning literally "after death". The investigation is started after detection of an event, such as a murder, robbery, kidnapping, hacker intrusion or fraud. Detection of illegal activity is thus a key premise for a forensic investigation. More recent trends move in two directions: Forensic readiness and proactive crime prediction.

Forensics readiness is about ensuring that potential evidence is collected and secured ahead of time [18]. It is motivated by the realization that it is often too late to secure the data after detection. Another important goal is to minimize the cost impact of the organization during an investigation by having solutions for data extraction without needing to shut everything down. There is always a trade off between restoring a system versus securing and investigating it which will lead to down time or lost evidence unless prepared for.

Crime prediction focuses on identifying predictors of feature crime and is especially interesting in terms of avoiding terrorism. It is controversial, both in terms of how effective it is and because it bypasses fundamental principles of requiring reasonable suspicion and probable cause for starting an investigation. An interesting debate highlighting these issues can be found in [19]. The data sources will typically be the same regardless of being proactive or reactive.

A forensic investigation has the potential for follow up as a legal pursuit in a court of law. Digital data can be used as evidence in order to link subjects to criminal activity. It is therefore important that data is collected in such a way that it is preserved in its original state and protected from manipulation.

Digital data can be user data, application data or system data [4]. It can be gathered via the network or from the devices hosting it [8]. Either from permanent or temporary storage. Digital data changes extremely rapidly when being processed inside a processing unit, is cached in memory or transferred on a network wire. Less so on a hard drive or on other forms of permanent storage. It is not practical or sometimes even possible to secure all data. A selection must be done and any intervention on live systems must be documented.

A chain of custody is started at the moment potential evidence is secured. It is a log describing who acquired it, where and how it was gathered. The evidence is then tracked until it is destroyed or returned. A major concern with physical evidence is manipulation before using it as evidence in court. Integrity of binary data can be verified using cryptographic hash functions mitigating the risk of manipulation. Other risks such as confidentiality breaches via missing access control and authentication [4] are still important issues.

Another important principle is repeatability. The general idea is to track the steps required in order to reproduce the evidence from the source media. Documenting from what media, what partition, in what directory, in what file and where in the file supporting or refuting evidence is found. One can argue that the means of discovery, be it by search, visually or by some automatic algorithm is of secondary importance as long as the evidence can be reproduced and verified using an alternative trusted methodology.

Theoretical frameworks [17] and ontologies have been created to structure the digital forensics process and how it interacts with the more general forensics science [20]. One such framework is illustrated in Figure 1.

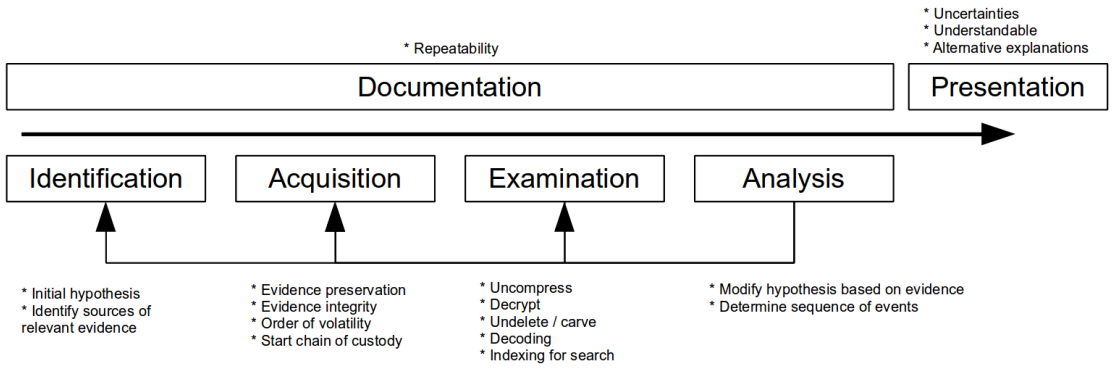


Figure 1: A model for digital forensics based on NIST SP800-86 [17] and descriptions in [4].

The process starts after suspicion has been raised. Relevant sources of digital data are identified and the framework ends with presentation of final conclusions. In between we have stages

focusing on preserving, preparing and analyzing the digital data. In a forensic investigation, digital evidence is just one of many sources of information. Other sources includes interviews, physical printouts, biometric traces and perhaps weapons. The stages still apply although methodologies might differ.

Acquisition deals with preserving the evidence by making a controlled copy of it. This step require prioritization in terms of what data to collect as it is impractical to collect everything.

Examination deals with how to transform digital data of various forms into human readable data. It can consist of uncompressing and decoding data. Decrypting or in worst case trying to decipher enciphered data. It is described as the *complexity problem* in [4].

Analysis is the art of putting puzzle pieces together into a story. Deal with inconsistent and missing data. It is hypothesis driven. A first hypothesis can be that X happened. What traces would X leave? Can we find these traces? If not, could Y have happened? Figure out what traces Y would leave and then look for them. A very common analysis method is to organize events by time. The power of it is chronology. An effect happens after an effect and not the other way around. Time can tell a story, but there are problems. The *unified time-lining problem* as explained in [4] deals with different timezones, clock skew, clock manipulation and various syntax inconsistencies.

2.2 Types of evidence

Computer crimes can be divided in 3 categories [21][22] and they all leave digital traces:

1. Where the information system is the target
2. Where the information system is an important tool in the crime
3. Where the information system is incidentally involved.

An adversary can attack the information systems directly with short or long term goals of exploitation, using both technical or social vulnerabilities. Digital traces from active attacks on computer networks can be logs of scan sweeps, denial of service attacks, access to restricted information and discovery of exploited vulnerable software.

High technology can also be the enabling or supporting factor of misuse. Examples are distribution of unacceptable content and planning of terrorist attacks. Perhaps even more important are traces left behind simply as a consequence of living in a digitized world [23]. Email, chat and Internet browsing history tell a story of social circles and interests. Meta data in photos can reveal when and where a picture was taken. Status updates on popular social networks contain time and location information and can give a detailed picture of a persons activities when combined with cell phone logs and credit card transaction details. These sources are getting increasingly more valuable in solving every day crime.

A famous saying in software security is that in order to breach security only one mistake is needed allowing and adversary to pass through the security mechanism. The same thing can be said for forensics: Forget to hide just one of many traces and your alibi could be broken [10].

A useful way to get an overview of digital crimes is to look at the organizations responsible for fighting digital crimes. Norway has several departments of relevance. Kripos, PST and Økokrim.

Kripos[1] focuses on organized or severe crimes. It is a part of the police, and tasks directly relevant for digital forensics are fighting Internet harassment, fighting child abuse, computer intrusions and content piracy.

Politiets Sikkerhets Tjeneste (PST)[2] concentrates on tasks such as terrorism, espionage, attacks on governmental bodies and dealing with weapons of mass destruction. The stakes are high and proactive steps are required. Targets under investigation actively avoid being compromised and hide their traces. Various surveillance techniques can be used, such as active monitoring of communications.

Økokrim[3] deals with economic and environmental crime. Economic crime can be related to taxes withholding, corruption and erroneously reporting assets during bankruptcy. The Enron case investigation was based on this last one. Other examples are stock related insider information, monopoly issues, misuse of subsidies and whitewashing of money.

Environmental crime covers the regulation of labor, the upholding of laws regulating safety and working hours. It also covers protection of art and cultural objects, illegal dumping of toxic waste and other misuse of the nature.

Økokrim primarily gather data post mortem from accused organizations or individuals. Sources of digital data gathered for investigation tends to be as a consequence of using technology, not a result of active attackers with deep technological understanding trying to masquerade every step. Email and office documents are among the most common sources of incriminating information [9].

[1] Norwegian National Criminal Investigation Service at `https://www.politi.no/kripos/`

[2] Norwegian Police Security Service at `http://www.pst.no/trusler/`

[3] Norwegian National Authority for Investigation and Prosecution of Economic and Environmental Crime at `http://www.okokrim.no/`

2.3 Visualization theory

Vision is the human sensory input with the highest bandwidth and we have evolved to be very efficient at thinking visually [24]. Restrictions in human short-term memory can be eased with visualizations since comparisons can be performed without performing context switches that rely heavily on short term memory [8][25, Chapter 2.3.2]. A visualization can also be auditory, taste or touch based but our limited bandwidth and processing capabilities for those sensory inputs makes visual visualization the most attractive.

Two interesting perspectives of visualization deals with how something not directly visible can be transformed to a mental image, and how processing power of computers can aid generation of these mental images

> "To form a mental image of (something incapable of being viewed or not at that moment visible)" [26]

This is a definition of visualization. Incapable of being viewed does not mean impossible to view, but that it is not directly visible for humans. Another definition:

> "Visualization is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Visualization offers a method for seeing the unseen." [27]

Human vision is limited to visible light which is one of many limitations in what humans can observe. The transformation from "symbolic to geometric" highlights how observations in the form of numbers, or theories in the form of formulas can be mapped to objects with shape, size, location and color.

The goal of visualization research is:

> "... to integrate the outstanding capabilities of human visual perception and the enormous processing power of computers ..."[28]

A similar quote on visual analytics highlights the same idea:

> "Visual Analytics is the integration of the outstanding capabilities of humans in terms of visual information exploration and the enormous processing power of computers to form a powerful knowledge discovery environment" [29]

Three main questions arise when data is to be visualized[28]:

- Why visualize?
- What does the data look like?
- How to visualize it?

Reasons for visualizing can be exploration of the data, hypothesis verification or presentation. The properties of the data are very important. A measure can be nominal, ordinal, interval and ratio. How to visualize depends on the type of data and the usage of various geometric transformations.

Visualization as a technique has been used by humans for thousands of years. One of the earliest versions were 2-dimensional maps of geographic locations [30]. The author of this cited paper walk trough the history and evolution of visualization techniques. The term *information visualization* is used as the broadest term, and he divides it up into *scientific visualization* and *data visualization*. Scientific visualization is focused on realistic rendering of physical phenomena as typically observed in fields such as engineering, physics, biology and medical care. Data visualization is separated as being visualized from abstract data.

Simson Garfinkel explains the need for visualization for both discovery and presentation in his 4th OSDF conference presentation [31]. What is needed are *data driven* visualizations as compared to *hand-drafted* illustrations that he claims is most common today. Data driven visualizations stays objective as far as the collected data is concerned, and what is needed is to spot trends, clusters and anomalies.

Ben Shneiderman summarizes guidelines for information visualizations as:

> "Overview first, zoom and filter, then details-on-demand" [32]

He explains a taxonomy of 7 data types from simple 1-dimensional lists of items, moving to 2-dimensional data such as maps and 3-dimensional real world object data (spatial data). The temporal domain is a fancy word for time and is special because it adds context such as beginning, overlap and end to the previously mentioned categories. The next three categories are abstract, even though they are rooted in the physical 3-dimensional spatial plus temporal world: Multidimensional dataset measure properties instead of location and can contain arbitrary number of *dimensions*. The last two categories are tree and network. They describe the connection between objects and the main difference is that a tree has as one link to a parent node while a network is connected arbitrary.

Overview first is a zoomed out perspective where details are suppressed. For 3-dimensional spatial data this can be showing the entire object, and for temporal showing the whole time period. *Zoom and filter* is the interaction allowing the user to narrow down on interesting things and hiding uninteresting ones. Zooming in on a 3-dimensional object will reveal more details and in a multidimensional dataset filtering can be focusing on objects with certain properties or ranges of values. *Details-on-demand* can be the result of zooming and filtering where enough visual area is left to display details that would otherwise clutter the display.

Three additional tasks: relate, history and extract are not included in the quote, but are equally important. *Relate* can be explained as a horizontal change of parameters while filtering and zooming in on data. It is especially interesting for *network* data where objects are naturally linked. Suppose you are looking at email messages for a particular person at a specific time period filtering on a key word. A related action can be to see the same zoomed and filtered view for the sender of one of the currently displayed messages.

Keeping track of the interaction performed is covered by the *history* task. It includes the ability to go back to a previous selection of the data and to replay steps performed. The *extract* task deals with export and saving of interesting discoveries made. A digital forensic investigator must be able to reconstruct evidence by documenting the progress, and both of these tasks relate to this requirement.

Edward Tufte has become an authority [33] in the field of data visualization and has authored many books combining theory with exemplification [34][35][36]. He explains that the essence of quantitative thinking is "compared to what?" [36]. Data should be normalized and standardized such as inflation adjustment of monetary values [34]. He gives examples of how Choropleth maps[4] should be colored by density, not as totals in arbitrary areas like country borders and counties [35]. He emphasizes data variation over design variation and uses emphasizes the term *small multiples*[35]. It is a term for many repetitive charts using the same axes, scale and orientation but with different data. This technique supports easy comparison without having to interpret each chart individually.

A visualization should also aid understanding by showing the data truthfully. Another principle is "to clarify, add details" [35]. Cluttered and confusing visualizations are not the fault of too much information, but rather a failure of design. He warns about the usage of area and volume for representing 1 dimensional data as it greatly exaggerate larger values compared to smaller ones [34]. Do not lie with scales and use clear labeling. Avoid moire effects and illusions, 1+1=3 effects as he calls them [35]. Use natural colors and be careful with high contrast. Remember that rainbow colors has no ordering and that some people confuse certain colors because of color blindness. Avoid chart junk meaning everything not representing or explaining the actual data. Commenting and marking interesting observations is still very valuable [34]. Lastly, do not make puzzles. If a specific PhD is required for interpretation of a visualization, then it probably will not be of much value to anybody else.

Tufte introduced two measurements *data/ink ratio* and *data density*[34] to give quantitative numbers on how well a visualization performs in terms of how much data it shows. The goal is to maximize them. Data/ink ratio is simply the ratio of ink used for data compared to all ink used in a graphics. Muting and minimizing of grids are examples of ways to increase the ratio. Data density is how many data points are shown per area of chart space, and can be increase by adding more data or by reducing the consumed area.

Date and time are important in forensics, but Tufte warns that chronological order most often does not explain causal relationships. The actual cause could be missing in the ordered events, and there can be many cause and effect relationships hidden in the data.

[4]Choropleth maps: Measurements color coded on top of geographical maps

Techniques for geometric transformations can be separated into 5 categories [8][5]:

- **2-dimensional and 3-dimensional plots** such as bar charts, scatter plots and volumetric representations in euclidean space. A good summary of methods can be found in [37] and a presentation on historic trends in [38].
- **Geometric transformations** where multidimensional data is transformed so to fit into a 2-dimensional or 3-dimensional space. One example of this approach is Principal Component Analysis where data is transformed using uncorrelated axis keeping as much variance as possible.
- **Icon based** where different properties of an icon is controlled by different variables. A known example is Chernoff faces where the size of eyes, mouth, chin, eyebrows and other properties are controlled by separate variables.
- **Pixel based** where individual pixels on a monitor are color coded and displayed as a function of multi-dimensional data. These methods leverage the high pixel count of modern displays.
- **Nested visualizations** where the dataset is partitioned and visualized in steps in a networked fashion. Interaction is needed to reach all data points. An example is browsing of file system folders. Sub folder content is visible once entered.

Visual data exploration is summarized in this framework[39] and illustrates the power of combining visual and formal modes. Examples of formal methods are pattern discovery, clustering, anomaly detection, classification and prediction. Visualizations can give deep insights into data, show how models work and guide parameterization of them. Mathematical models can automate knowledge discovery on similar data. This is shown in Figure 2:

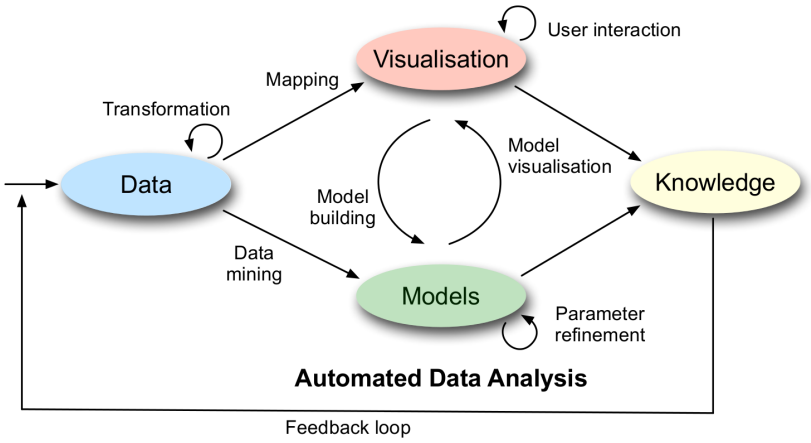


Figure 2: The path from data to knowledge can go via manual visualizations or formal models. The synergy of combination is shown in the middle. Illustration from [39].

[5]The original source is by Daniel A. Keim. "Datenvisualisierung und Data Mining"

2.4 Anomaly detection methods

Humans are very efficient at spotting anomalies when supported with effective visualizations, but it requires focus and manual labor. A more attractive solution is to have a computer to do preprocessing of the data. Computer algorithms can mark candidates of suspicious activity that in turn is left to human interpretation.

Motivation looking for anomalies is based on guidance given during digital forensic labs: Look for the unusual and follow the leads [40]. A concrete example during investigation of a compromised web server was the size of certain log files. Looking through the content revealed thousands of lines from a single IP source with an unusual user agent string belonging to a vulnerability scanner. This knowledge revealed hints of an IP address together with multiple key words to guide the remaining investigation.

An outlier is a synonym for an anomaly and can be defined as:

> "an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism" [41]

When malicious software is performing an action on behalf of a user, then it is a different mechanism. Perhaps similar changes in mechanism can be observed in email.

Computational forensics [1] includes the usage of machine learning techniques for aiding the forensics process. Machine Learning [42] can be defined as having an algorithm automatically extract knowledge from data without explicit programing. If the machine can improve it is performance at a defined task given examples then it is said to be learning from the examples. If a problem and performance at solving the problem can be clearly defined, then calculating optimal solutions is basically an optimization problem where analytical or heuristic search methods can be applied.

There are many machine learning algorithms, and they can be categorized into two modes of operation: *Supervised* and *unsupervised*. In supervised learning, preclassified examples are given to an algorithm. The algorithm learns a model based on training data, and the effectiveness of the model is measured by testing it on unseen examples. The goal is to generalize in order to maximize the probability of classifying an previously unseen examples correctly later on. Generalization is a balance between having a too simplistic model (under-fitting) and a too complex one (over-fitting) where decision borders match the training examples perfectly, but fails on novel examples.

In unsupervised learning no classification information is available and it is up to the algorithm to extract knowledge from the measurements alone, typically by locating patterns. Examples of methods in this category are clustering and certain anomaly detection techniques.

There are two very important theorems: One regarding choice of algorithm and the other regarding the number of measurements. The first one is the *no free lunch* theorem [43]. It basically states that if a method performs well on a set of problems, that means there will be problem areas where it performs poorly. On average no algorithm is better than any other. Performing good in one area means it is biased here. There is no such thing as the universally best algorithm. It is always an adaption to the environment. If evolution can be called an algorithm then it might be an exception by definition. Adapting the method to the environment is needed.

The other theorem is named the *ugly duckling* theorem [44]. It states that two distinguishable objects get more and more alike the more things about them is observed. In other words: difference is a function of biased observation. A swan is prettier than an ugly duckling because the observer prefers certain features over other. The same is true when trying to detect malicious software or suspicious email. All properties we can measure will not be useful in the wanted separation.

Data must be collected and preprocessed independent of machine learning mode. Collection is to measure and digitize physical observations into machine readable form. In an ideal world we would know exactly what things to measure for a given problem, it would be easy to measure them and there would be no noise.

The realistic strategy is often to guess or measure things that are easy to measure. An example of task is to find incriminating information in email. Easily available measures are message size, the subject string, the timings and the number of receivers. But is size of the message really an indicator of potential evidence? Potential evidence is a loose definition but it could be narrowed down a case of proving distribution of insider information. Size of messages could be relevant if the information is a huge attachment. But it could equally likely be a short textual message.

A more likely candidate field is the subject, but it is simply a list of ordered numbers representing characters used when displayed to human operators. Text must be converted into something a machine can make use of in order to be a useful feature.

The result after the collection phase is typically a set or a stream of observations each containing multiple measures. The term sample, example or point is often used for the observed object. The individual measures are attributes or features. This is illustrated in Figure 3.

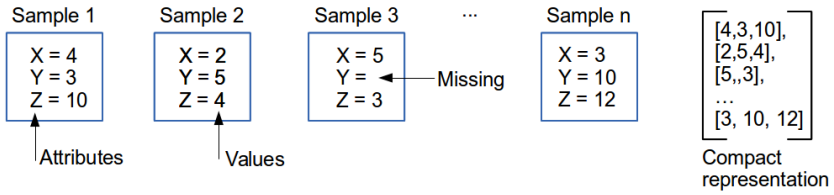


Figure 3: Each sample has multiple attributes with individual measurement values

The take away is that domain knowledge and scientific experimenting is needed in order to find an optimal method and way of features for a given problem. Methods exist for removing irrelevant and redundant features [45] but they can not suggest missing ones.

A survey on anomaly detection [46] uses the following definition:

> "finding patterns in data that do not conform to expected behavior" [46]

Similar fields of study are novelty detection and noise removal. As the author explains, the principle of novelty detection is to trigger when observations never seen before are discovered. These observations are treated as normal once discovered. In the field of noise removal, outlying observations are considered unwanted and filtered out. Defining outliers could be done by first defining all situations of normal. Problems with this approach is that there might not be a clear borderline where an observation turns from normal to an outlier. It can be gradual. Often adversaries try to hide their actions, the different mechanism, by making traces left behind look normal. It is also sometimes very difficult to define what normal is. What is normal can evolve over time and actual noise can be mixed in with the data. An example for email is spam.

Anomaly detection methods can be categorized in 3 categories [46].

1. Point based: Each data sample point has behavior information. It is treated independently while determined normal or abnormal as compared to all other samples globally.
2. Contextual or conditional based: Each point has contextual information in addition to behavior information. Examples of context can be membership in a category, a location or a time period. A point with the same behavior could be normal in one context and an anomaly in another.
3. Collective based: A combination of points is abnormal when observed in an unusual order. An example could be sending no messages for a long period, even though sending no messages a single day is not unusual. The author points out that points in such datasets must be related [46].

Input of anomaly detection methods can be fully supervised where examples of both normal and abnormal points are given. It can be semi supervised where only one of the classes is given. Typically the normal class. It can also be unsupervised where no classification is given. Points can consist of different types of data: categorical, binary and continuous values. A mixture is not uncommon. The output can be binary yes or no, or scored so that the points can be presented as *top n* anomalies.

A lot of research has been done on point based methods. Categories of methods summarized from [46] are:

1. *Classification based*: This is a supervised method where normal and anomalous points are prelabeled. The classes can be one or many and the learning phase tries to learn what these classes look like in terms of decision boundaries. The output of classifying an unseen sample will be normal or abnormal. Examples are one class support vector machines, Bayesian networks, neural networks and rule based variants such as decision trees. Access to preclassified realistic examples of all normal or abnormal patterns is a major limitation.

2. *Clustering/distance based*: The assumption is that anomalies does not belong to a cluster, or that they lie far from their allocated cluster center. Outliers can also be defined as small or sparse clusters. Examples of methods are self organizing maps and k-means. A distance measure is needed for clustering, and a metric is typically used. Requirements for a metric is a distance 0 or greater between points, that the distance is the same in both directions and that the distance to itself is 0. A fourth criteria is that the distance between two points is equal or shorter than going via a relay point. Euclidean distance is common.

3. *Nearest neighbor/density based*: These methods are also based on distance. The difference is that they only care of the points in the local neighbor instead of the global clusters. Methods can be to count the distance to the k nearest neighbors or count the number of neighbors within a fixed radius. Partitioning of the data domain is a known method for speeding up computation time.

4. *Statistical based*: These methods are based on stochastic models and outliers are in areas with low probability. A classic example is classifying observations more than 3 standard deviations from the mean as outliers. Other methods are classifying observations more than 1.5 the distance from the median to the 1st and 3rd quartile as outliers. Other methods include regression where a function is fit to the data, and points far from the predicted function output at that location are classified outliers. A major issue in using statistical method is making wrong assumptions about data distribution. Arbitrary distributions can be approximated using histogram based methods where the frequency is sampled in bins. A limitation is that each attribute value can be normal but highly unusual in combination.

5. *Information theoretic based*: A complexity measure such as entropy (measured in compressibility) is defined. The goal is to find a subset of the data with minimal complexity while at the same time keeping the number of items removed low. The removed points are the outliers and the rationale is that the outliers add more to the complexity than normal points. It is a dual optimization problem with the dangers of getting stuck in sub optimal solutions or alternatively an exhaustive search through all possible subsets.

6. *Spectral methods*: These methods try to express as much variability as possible using dimensionality reduction techniques with the goal that outliers are more significant in the reduced subspace. One example of such projection is principle component analysis.

There is no guarantee that what is considered evidence in a particular case will have outlying characteristics. Most outlier detection methods will always give an output, even when data supplied does not contain anything abnormal as determined by a human. What is needed is to understand why certain outlying patterns would have a higher probability of predicting suspicious activity. A contextual based method is an interesting approach for email investigations. Each email account can be a context variable, a group of users can be a context variable and there might be other ways to define context. The point based methods can be applied a contextual settings if "analyzed with respect to a context" [46], meaning doing anomaly detection in subsets.

Requirements for tools in computer security are different than in forensics [13]. Low false positives, real time output and autonomous function are key factors in computer security. Forensic investigations are intensive in human intervention. Real time constrains are relaxed. Algorithms with high computational complexity can be considered for automatic clustering and categorization of documents. Algorithms that can adapt to feedback from the investigator on a per case basis would be beneficial [13].

Tuftes principles in visualization is not to create puzzles [34] and the same applies here. The output and function of algorithms must be explainable to a court [47].

2.5 Related works

The intention of this section is to give a broader overview of visualization techniques used to support digital forensics investigations. 3 focus areas are defined [47], being **differential analysis**, **streaming data analysis** and **social network analysis**. The first two covers digital forensics in general while the third narrows down on email related investigations.

Examples are numbered for each of the 3 subtopics. Examples not focused on visualization have a title beginning with *explanation*.

2.5.1 Differential analysis

File systems have traditionally been in focus in computer forensics. Timestamps from various sources from the file system and the files within are collected and presented in timelines in order to answer what happened at a given time. Aggregation of low level events into high level actions has been also been studied [48]. Another interesting approach is comparison of different sources, directly or against previous snapshots. The commonality is differential analysis [49][50]

Example 1: Cyber Forensic TimeLab

Cyber Forensic TimeLab [51] uses *small multiples*[35] for easy comparison of events across various time sources of resources on a computer. The tool extracts timestamps from a harddrive using the FAT or NTFS file systems. It will find timestamps in JPEG images, mbox email archives, Windows messenger chat logs, Windows system logs, the registry and link files. All of this information is displayed in small histograms categorized by type as shown in Figure 4.

Figure 4: Spotting co-occurring events is easy using this visualization. Illustration is from [51].

Another interesting technique used in the paper is dynamic loading of file preview in their hexadecimal view. A file can be very large and loading the entire file into memory before displaying the bytes will consume that amount of space. Instead they load the required pieces once needed while scrolling.

Example 2: Perspective wall

A similar technique is the perspective wall [52] where the authors used the modification timestamps of files, categorized by file extension. The visualization (Figure 5) uses a 3-dimensional perspective trick where the center of the timeline is right in front of the user, and the sides disappear back into the screen at an angle. This gives a sense of perspective. This technique is known as *bifocal display* [53].

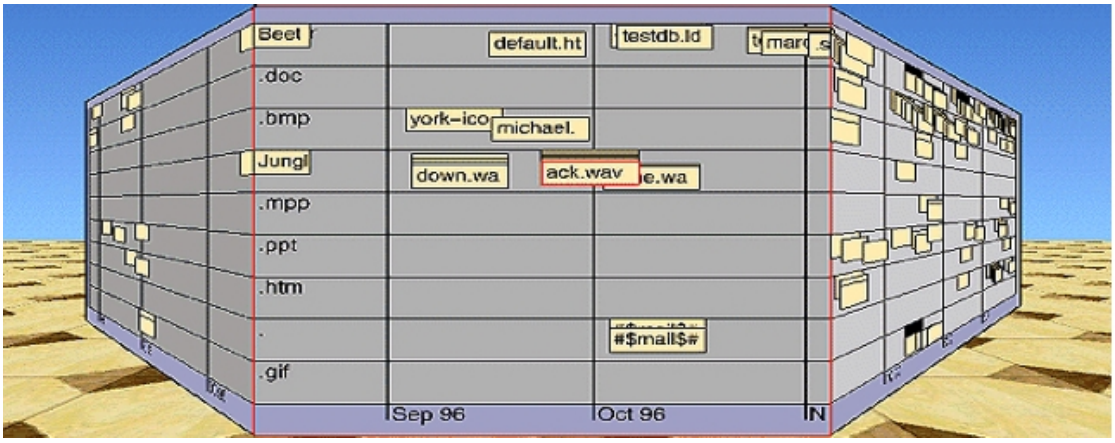


Figure 5: Notice how items far to the sides get compressed in width. The illustrations in the original paper are of low quality. This illustration is from [54].

Explanation: Differential analysis strategy

Garfinkel et. al. [50] explains a strategy for how to perform differential analysis based on experience of writing many such tools. The important takeaway is to describe the formal rules for creation, deletion, moving and copying. Creation is when an object did not exist before but does now. Deletion is when an object used to exist, but does no longer. Moving is when an object exist but is in a new location. Copying is when an object exist in more locations than before.

Differential analysis can be applied for malware analysis and user monitoring. The main idea is to take snapshots at different times and create delta images describing the change from one snapshot to the next. In malware analysis this technique can drastically reduce none-relevant information, given that the malware was introduced after the first baseline. This technique is commonly known from different revision control systems such as Subversion and GIT[6].

[6]Revision control is commonly used during software development in order to track changes to the program code and for synchronization of code between developers.

Example 3: Volume shadow copy

Differential analysis works when you are in control of the system under investigation, but what if you work post mortem? Later Windows operating systems have a function called *Volume Shadow Copy Service* introduced in Windows Vista. It can be a window back in time allowing the study of file changes over time for a volume on a hard drive. This shadow copy functionality is studied and visualized in [55] where the authors developed a prototype called *Change-link*.

The assumption is that changed data is more important than static data regarding what occurred on the system. Change-link takes the available snapshots and calculates delta changes between them. Figure 6 shows how the program looks like from the paper description. This technique is naturally limited by operating system, space assigned to the shadow copy service and whether or not the service is enabled.

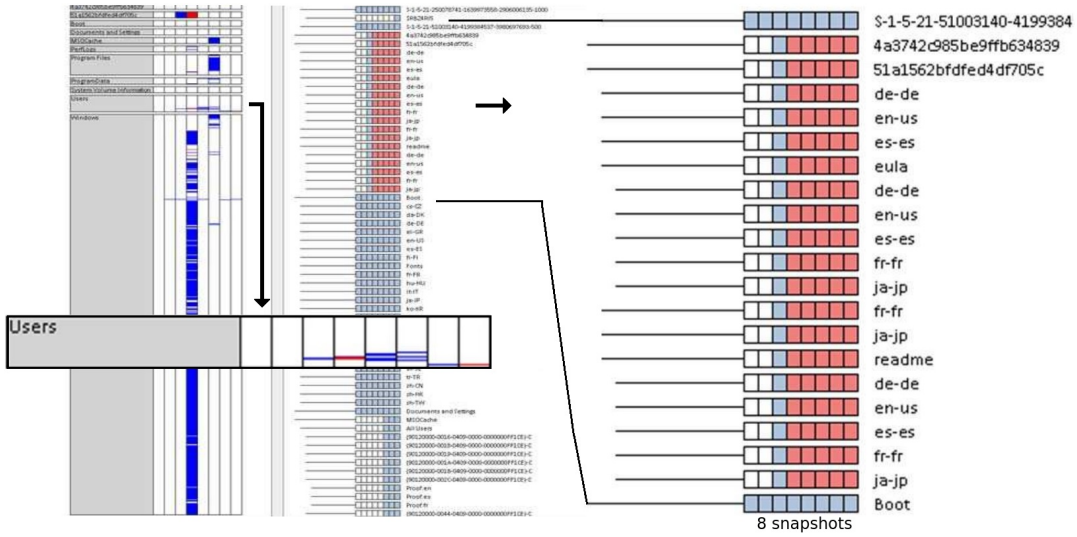


Figure 6: The left system root view shows deleted files with red lines and created files with blue lines. The usage of *small multiples*[35] makes it easy to see how much change occurred and at what snapshot it happened. The right view shows sub folders and depends on selection to the left. It shows file not yet created as white, existing files in blue and deleted files as red. The length of the prepended line is proportional to the folder depth. The illustration is from [55].

Explanation: Malware cross correlation

This example does not focus on visualization directly, but highlights important principles regarding feature selection in *unsupervised* machine learning. Distance based clustering in this case. It is also relevant because it combines data from multiple infected computers.

The topic was identification of botnet malware[7] infections and was published as a master thesis [56] and a paper [57]. It was based on cross correlation using simulated data from multiple computers, and the hypothesis was that file traces of malicious software would be easier to discover when clustered together.

The data used was primarily file system meta data like file timestamps, size, allocation status, path, identifiers and permission. It also calculates MD5 and SHA1 hashes used for removing known files. The reduced set of unknown files was scanned for additional text information such as IP, email and URL addresses. Content entropy was calculated and file headers were fetched using the Unix *file* command as these are more reliable than the file extension.

Each file is represented with these measurement attributes. File meta data from multiple computers were merged after adding administrative attributes such as source computer. Data was clustered using a method called k-means. K-means is distance based and the most common distance measure is the Euclidean distance which means it requires numbers.

Many of the attributes are not *ratio* numbers. Examples of such are the time strings, path, permissions, IP, email, URL and file type. The machine learning tool *Weka*[8] was used for clustering and it has automatic conversion for treating such attributes as nominal values. A simple explanation can be illustrated with an attribute such as permission: It contains the possible values *read*, *write* and *execute*. They are separated into 3 new attributes where the presence of that value in a particular instance is given a binary 1 (true) or else a 0 (false). It is questionable whether this is a good approach for a given attribute. Another important question is whether all measures should be included.

[7] Malware is malicious software and a botnet is a group of infected machines used for distributed denial of service attacks or for monitoring of the computer owner. Misuse can be stealing credit card information or login sessions

[8] Weka: Data Mining Software developed in Java at `http://www.cs.waikato.ac.nz/ml/weka/`

A botnet was installed on five 5 machines using virtual machine technology, all cloned from the same baseline. Hash filtering were done using this baseline. Case knowledge such as known IP-addresses and timestamps were used when the author analyzed the clusters manually after clustering. *Weka* built-in tools were used to show interesting patterns visually as seen in Figure 7.

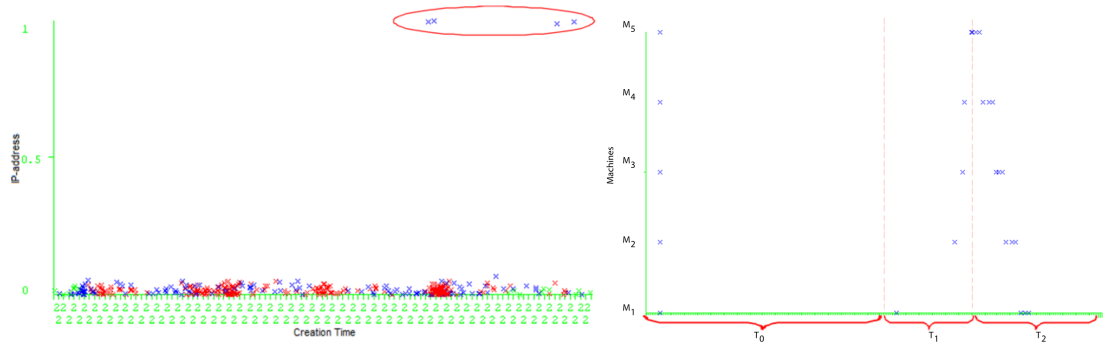


Figure 7: On the left we see the creation time versus the file having an IP-address string or not. It seems most files does not have one. At the right we see access time versus machine id for one of the clusters, filtered by having a particular IP-address somewhere in the file. The order of infection is easily visible. Illustration from [56].

Between 431 and 434 files were left on each of the 5 clones after hash filtering with the baseline. The count before filtering was approximately 13.871 which means 97% of the files were removed. This is far more than public hash databases would be able to remove. This is due to the fact that the baseline hashes include custom files that would normally be different between installations but just happen to be identical because of cloning of a baseline. The experiment was thus cross source correlation combined with baseline differential analysis techniques. This level of filtering is thus not realistic for sources of evidence acquired outside of controlled environments where such a baseline is available. Another important observation is that the malicious files were not grouped together in dedicated clusters. They were often grouped, but mixed with all the remaining files having similar properties and it is unclear how much benefit can be associated with the clustering compared to searching for known IP-addresses and timestamps directly in the meta data.

Example 5: Self organizing maps

This example uses a clustering method for visualization of data. It is a good example of trying to combine analytical support with visualization in order to get a quick overview of the evidence, but it also exemplifies the dangers of abstract visualizations.

The example is from a master thesis [58] from 2007 that studied usage of self-organizing maps[9] for investigating file meta data of pictures, music files, temporary Internet files and proxy logs. The attributes file type, size and creation date were extracted from the forensic tool *Forensic Toolkit (FTK)*[10]. It was preprocessed by extracting *day of week* and *time of day* for dates. This data was visualized using a tool called *SOMFA* developed by the cited author.

The self-organizing map maps multidimensional data down to typically 2-dimensions while still preserving the topology of the original space. Close points in the original space are close in the projected space. It builds on neural networks where each attribute is connected to an input layer node and a 2-dimensional grid of output nodes are connected to all input nodes. The size of this grid is a free variable. All samples are run through this network in multiple rounds. For every sample there is a fight between output nodes. The winner and neighbors get their attribute weights updated. The end state of the output nodes is visualized, typically as hexagon cells for each of the output nodes in the grid. The color of each hexagon is determined by one of the dimensions for the component map mode. An alternative is the frequency map where the color is dependent of how many input examples have been mapped to a particular output node. A few examples from the thesis is shown in Figure 8:

[9] Self-organizing map (SOM), also known as Kohonen map

[10] AccessData: Forensic Toolkit® (FTK®) at `http://www.accessdata.com/products/digital-forensics/ftk`

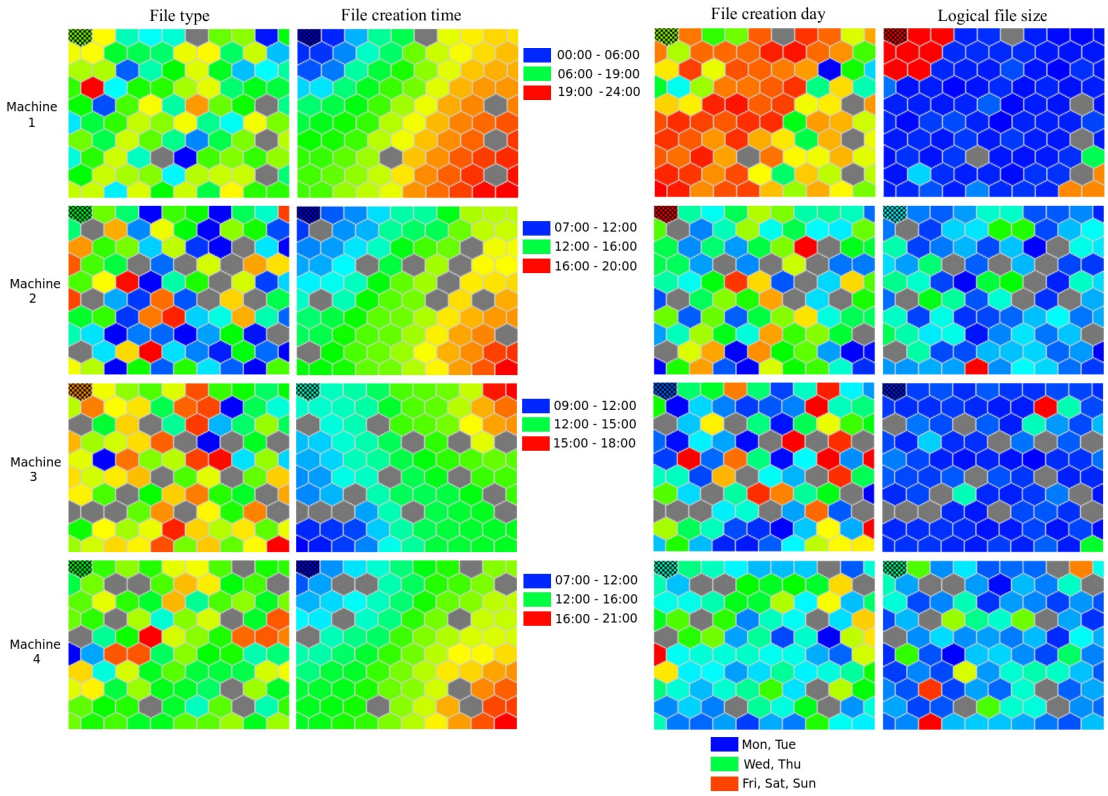


Figure 8: Visualization of file system meta data using Self organizing maps. This illustration is based on illustrations from [58]. Extra information has been added and the individual maps have been aligned for easier comparison.

The approach has a few problems: The first one is the choice of rainbow colors for gradual changes which is warned about in [35]. The result is that detailed legends are required in order to mentally map green, red, blue and yellow to ranges in time or file size. Another problem is that the color range vary between different datasets. The range is determined by the range in the current dataset, not the full range. This is especially important for fixed ranges such as *day of week* and *time of day*. This results in tedious comparisons as shown in Figure 8. These kinds of visualizations are definitely puzzles. Knowledge can be extracted from them with enough training, but the question remains whether this approach is useful for the average investigator.

Example 6: Webscavator

In contrast to the last visualization puzzle, Webscavator [59][11] is an example of a tool for analyzing web history. It is based on web technology and uses various time and word based visualizations. Plotting the *hour of day* as a function of day makes it easy to spot trends. At least three trends can be seen in Figure 9: Nothing happens during Saturday and Sunday. All activity is between 09:00 and 18:00. Close to no activity between 12:00 and 13:00.

A word cloud is a visualization technique where arbitrary long character strings are split up by word separator characters such as space. These parts are counted and the relative frequency determines size, color and location for each particular substring.

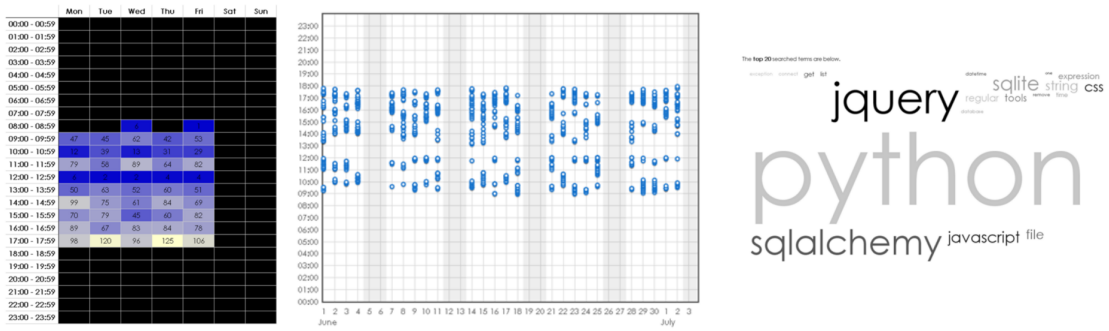


Figure 9: From left: Time of day heat map, time of day calendar plot and word cloud. Illustration from [59]

[11] Webscavator at urlhttp://webscavator.org/

2.5.2 Stream of events

Log files are generated locally from programs and the operating system. Web server logs is an important one. They are also generated by external devices such as routers, firewalls and intrusion detection systems. They all have in common that new events are generated regularly in a constant stream of events.

Example 1: Protocol identification by timing

Internet traffic is sent in packets and protocols could be determined by analyzing packet size and packet timings, even when the content is enciphered [60]. The authors plot packet size as a function of time for various protocols in both directions revealing visible differences. They also experiment with inter-package timings and plot the relationship between pairs of packet size. The last and the current one. This is illustrated in Figure 10.

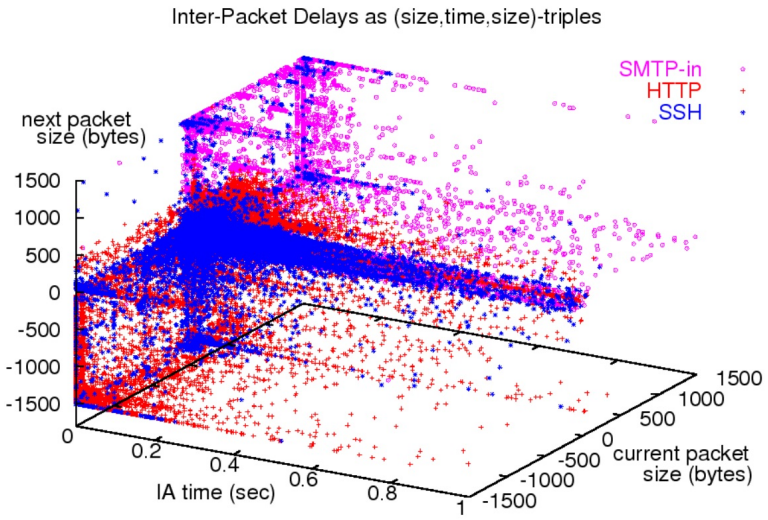


Figure 10: Difference in packet patterns are visualized, colored using the known protocol. Illustrations from [60].

Example 2: Visualization of TCP dumps

TCPflow[12] is a tool for capturing network traffic and it has the capability of visualizing network capture data. The author Simson Garfinkel [61] discusses the problems of scale when large values dominate smaller ones, and how using *broken axis* is a better than using logarithmic scales. He also explains design choices of the visualization module for TCPflow. It creates histograms of activity grouped by port, shows top source and destination addresses and distribution for ports independent of time. Graphs are split on incoming and outgoing direction. See Figure 11:

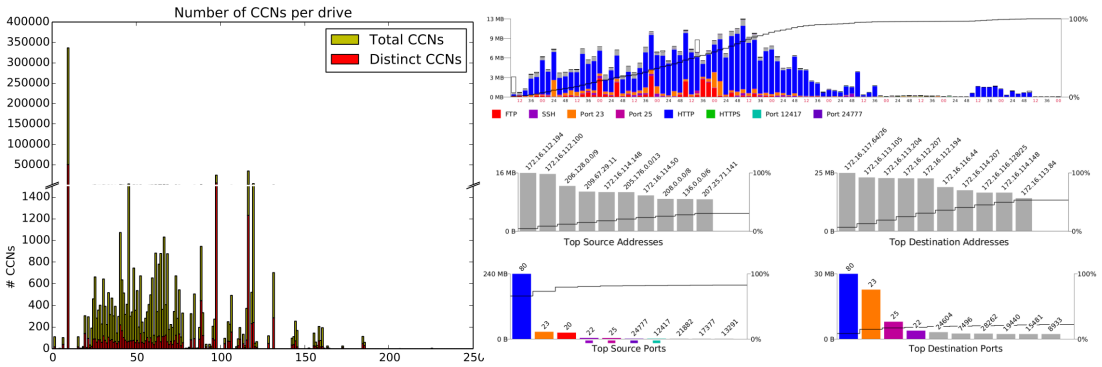


Figure 11: Left: The broken axis technique using two plots with different zoom.
Right: The TCPflow visualization showing key parameters such as port and address statistics. Illustration from [61].

[12]TCPflow at Github: `https://github.com/simsong/tcpflow`

Example 3: Visual insight into a classifier

This example is highlighted because it uses visualization to bridge the gap between a simple classifier and the human operator. In his PhD thesis [62] Axelsson argues that the relevant measure for effectiveness of an intrusion detection system[13] is not how good it is at detecting malicious behavior, but how well it can suppress false alarms. He introduces the Bayesian detection rate: The probability of an alarm being an actual intrusion as a better measure.

The bases for his work is access log data from a live Apache server at the campus where he worked. These logs contain the HTTP GET request, which is essentially what the user typed in the browser address bar.Duplicate requests were removed and the remaining requests cut into its constituent parts as illustrated in Figure 12:

URL: http://subdomain.domain.tld/**folder/file?id=15&color=red#part2**
Log: GET **/folder/file?id=15&color=red#part2** HTTP 1.1
Split into: «GET» «folder» «file» «id» «15» «color» «red» «part2» «HTTP 1.1»

Figure 12: Split symbols could be: / ? # + : = &

These substrings were counted. Some substring were very frequent, so a cutoff were set discarding the most frequent. The original request strings were scored based on their average frequency of substrings. The result is that the least common requests are scored the lowest. Directed edges were created by traversing the lowest scoring requests. This data graph was visualized using a 3D plot in the program Tulip[14]. Requests were classified into several attack classes by manually filtering out branches as they were inspected. See Figure 13 for what this 3D model looked like.

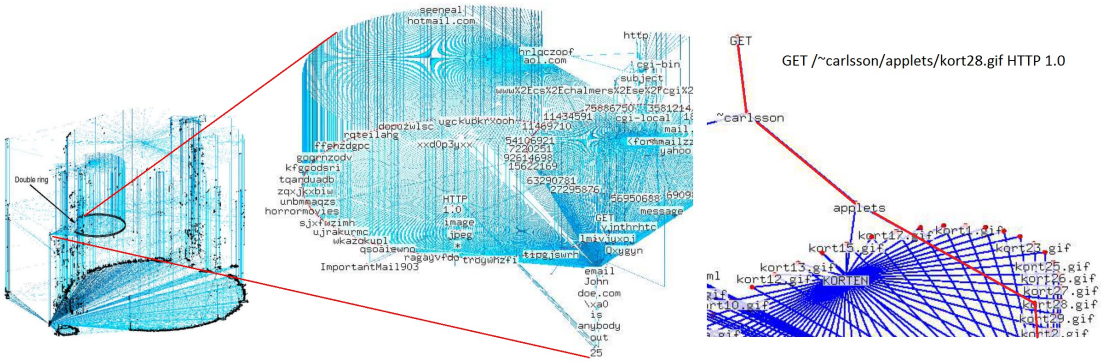


Figure 13: Individual page requests are broken down, appended to a graph structure and visualized in 3D. Illustrations from [62], modified with explanatory lines.

The next step was implementation of a *supervised* probability based classifier, inspired by spam filtering algorithms. An interface was developed allowing each unique request to be selected and

[13] Intrusion detection systems try to detect intrusions in real time. These systems monitor huge amounts of normal behavior a few malicious patterns hide.

[14] Tulip: Analysis and visualization of relational data, at `http://tulip.labri.fr/TulipDrupal/`

classified as either benign or an attack. Doing so would update the probability of the individual substrings to be benign or attack. Each request was scored based on its individual parts and classified based on a threshold. Each substring in every line of request were color coded using a heat map technique using the range red-yellow-green. Looking at a request classified as attack would immediately give the operator insight into why. See Figure 14.

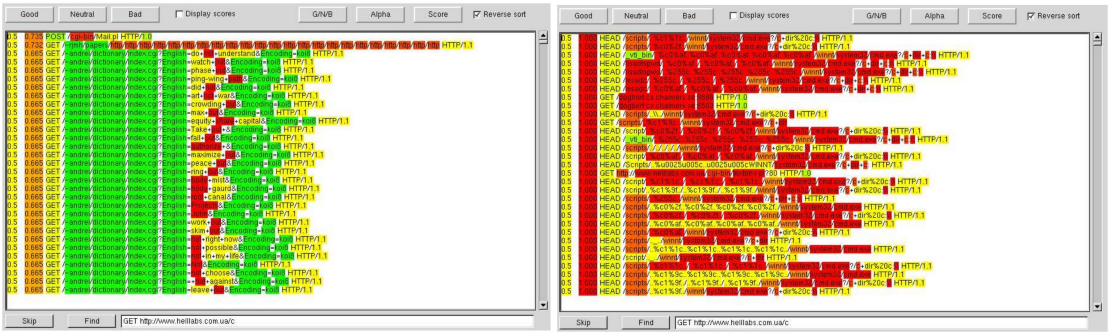


Figure 14: The probability of being benign in green and malicious in red. It is easy to see what individual words contributed to the classification. Illustration from [62].

The classifier is in essence an advanced form for white- and blacklisting of strings. A more advanced sliding window version was implemented later with better average performance. It is seen in Figure 15. A very important takeaway is that this method is for experts. A lack of expert knowledge of the underlying data can not be overcome by any visualization. Visualization can aid pattern discovery. Outlier detection can notify on suspicious patterns, but these methods can not interpret the meaning by themselves.

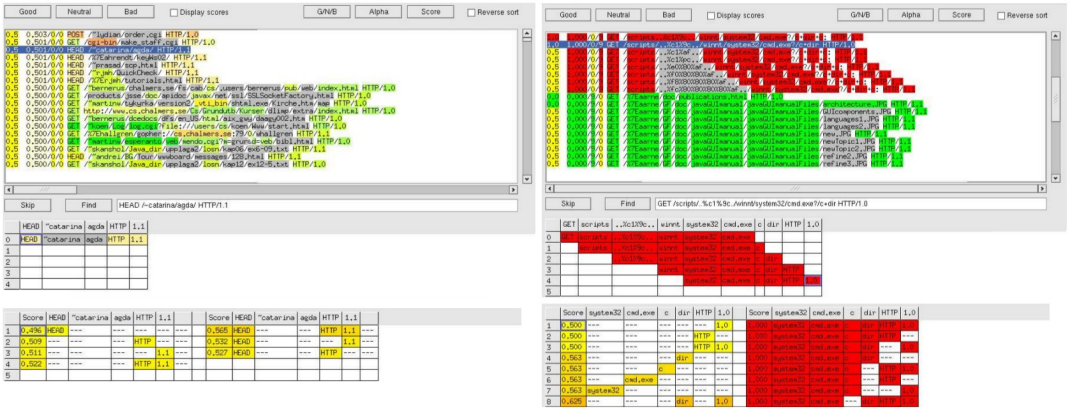


Figure 15: The addition of a sliding window makes the visualization more complex and harder to understand. Illustration from [62].

2.5.3 Social network data

Social network data is similar to streaming data, but is special because it directly reflects human interaction. Email and other social network data such as chat logs, Twitter feeds and Facebook posts all have meta data telling a story of relationships. When, how often and from where the participants communicate.

3 commercial solutions are also briefly mentioned. They have a title starting with *commercial*.

Example 1: Enrich email with external data

The content of an email message typically contains text. The authors of [63] developed a tool called Ontea as seen in Figure 16. It can recognize various parts of email text such as telephone numbers, names, addresses and dates. Lookup of recognized elements are used for enrichment of the original message with links to maps, phone book lookups and calendar suggestions.

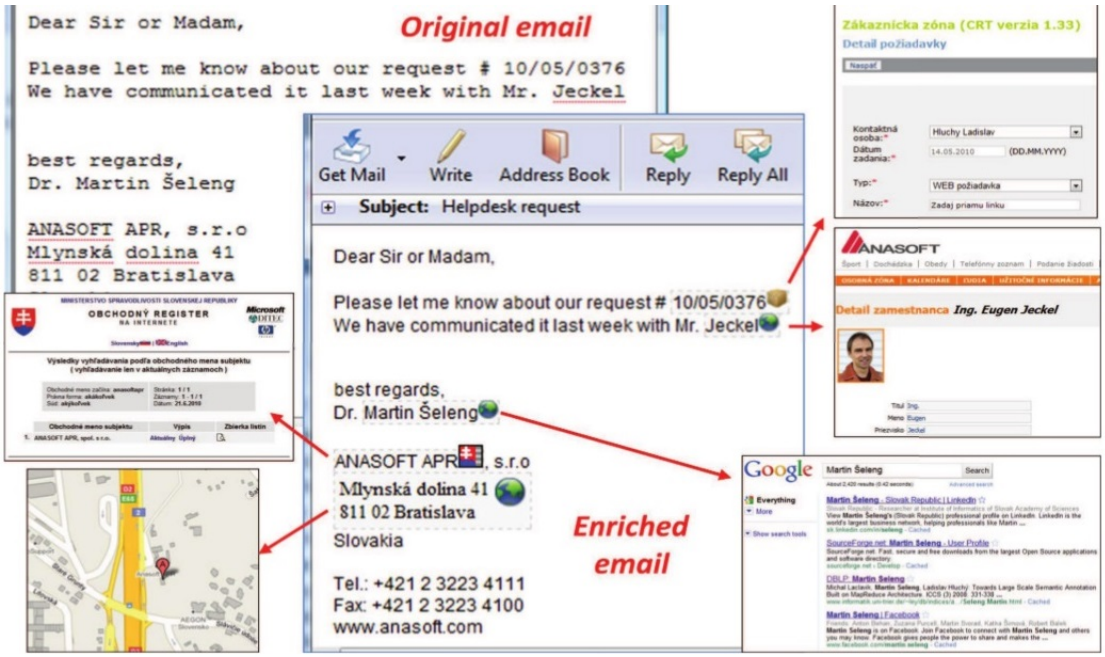


Figure 16: Notice how the original email has been enhanced with recognition of tracking numbers, persons and addresses. Illustration is from [63].

Explanation: Author identification

Email can be used for sending inappropriate content and techniques for automated authorship identification can aid investigations in cases where the claimed author denies involvement [64]. Examples of misuse are leaking of information, harassment and threats. Linking messages to its author(s) is called *author attribution* and is based on knowledge of earlier written content. Characteristics such as layout (structural) and language (style markers) could be used, but the question is how reliable these are and how they might change over time. It is essential to ensure that linking is based on characteristics of the author such as writing style, and not arbitrary factors like topic or message length.

The authors in the cited paper experimented with email messages from 3 authors on 3 topics. Number of messages ranged from 3 to 25 for each category per author. Quoted reply text and signatures were removed. Training data consisted of both layout and language-related features.

- Language-relates features were distribution of sentence and word lengths, the number of short words, function words[15] and a measure for vocabulary size.
- Structure related features were also used such as HTML tag distribution and number of attachments.

A supervised machine learning algorithm called Support Vector Machine (SVM) was trained and it was found that language attributes dominated, which means they were most important. The F-score on the language related test data ranged between 75.0% and 88.2% which means the probability of classification mistake is quite high. The results exemplify the importance of scientific testing for determination of error rate as demanded by the *Daubert standard*.

[15] Function words are words with little or no meaning by themselves

Example 2: Enronic

Text can also be clustered and classified into topics based common words using natural language processing algorithms. Jeffrey Heer discusses the dangers of such automated processing in terms of accuracy and faults, but suggests to use the classification results in an exploratory context using visualization[65]. He developed and demonstrated a system called Enronic for visualizing the output of one such clustering algorithm. The input in a preprocessed version of the Enron dataset already classified into various topics. It is split into a graph visualization and a message view.

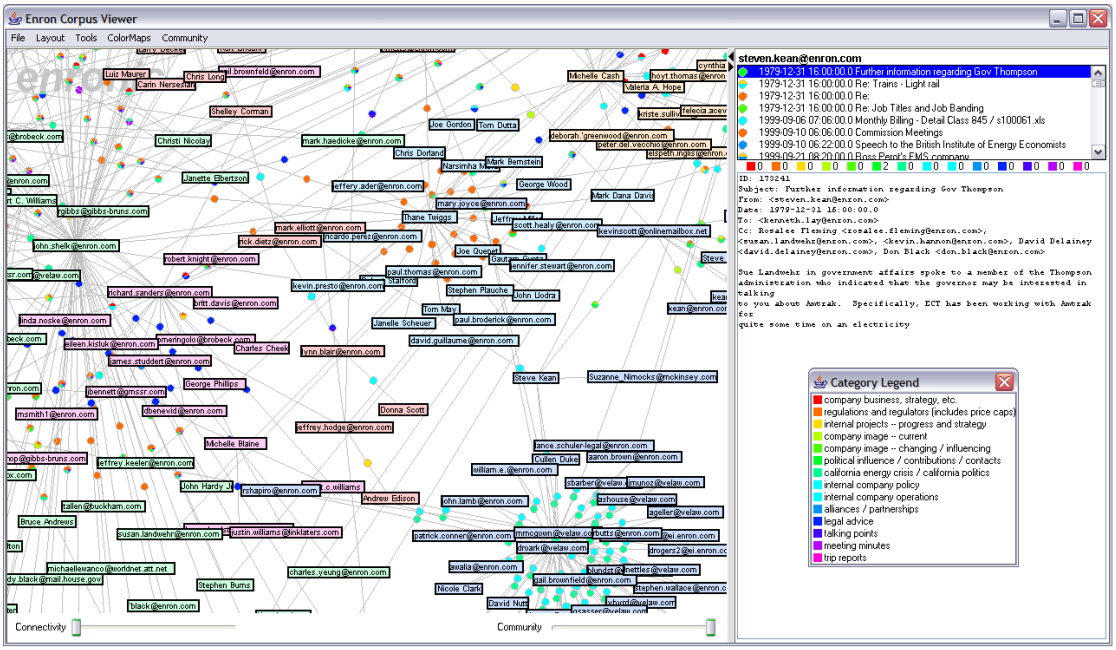


Figure 17: Natural language processing combined with interactive community clustering. The connectivity slider filters nodes at a certain connectivity level. The community slider clusters based on topic category using a slightly modified *Newman's algorithm*. Illustration is from [65].

The graph view (Figure 17) plots direct communication between accounts. Lines also have a pie chart representing the ratio of the various topics for that stream of messages. The graph visualization can be zoomed and it has controls for varying the level of minimal connectivity and amount of visual topic clustering. Clicking on a line in the graph view changes the content in the message view at right.

The message view has a list of individual messages color coded by the individual classified category and clicking any of these messages brings up the original headers and body text.

Example 3: Roles of people in conversations

Threaded conversations is a concept of nested messages for supporting discussion. Typically a question or comment is given by a person followed by replies. Either to the original author or to other replies. Threaded conversations is a common way to organize message boards. The reply functionality in email clients typically keep previous messages in the message body and creates a similar although not enforced structure. What role the contributors of such systems have is the question asked in [66]. The authors enumerate several interesting questions. A simplified version is given here:

- Who are the core individuals in terms of question askers, discussion starters, commentators, experts and administrators.
- How are core individuals connected? Any subgroups?
- What effect do new administrators or the loss of a member have on the interconnectivity of the community?

Reply networks are graphs where the persons are connected to each other and the edges are strengthened every time a person A replies to person B. *Affiliation networks* are graphs where authors are connected to threads, or more generally thread topics. Authors are classified in categories like answer people, question people and discussion starters. This is based on their *out-degree* and *in-degree*, how well they are connected with other active members and how well their neighbors are connected. Out-degree is a measure for the number of messages sent by an author and in-degree for received messages.

NodeXL[16] was used to render the graphs as seen in Figure 18.

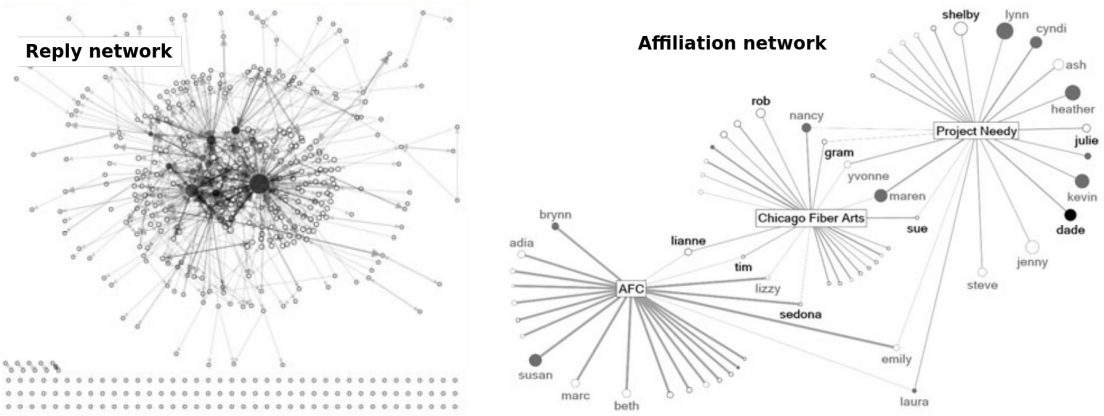


Figure 18: Left: Direct reply network based on data from an email list. Right: Affiliation network based on message data from discussion forums. Illustrations from [66].

[16]NodeXL: Network Overview, Discovery and Exploration for Excel at `http://nodexl.codeplex.com/`

Two additional centrality measures are explained in [67]:

- *Closeness* is the average number of steps (edges to follow) in order to reach all the other nodes, at least the accessible ones. The lowest scoring node is the most central one.
- *Betweenness* measures the probability of a node to be in the shortest path for other pairs of nodes. The highest number is then most central.

Example 4: Email Extraction Tool

A similar approach is found in [16] where the authors try to identify key actors, strong ties and bridges between groups. They built a tool called Email Extraction Tool (EET) and used an individual email account as input.

Email has fixed header field but can also contain headers quoted inside the message body. A design goal of EET was to mine these addresses so that a larger graph of forward and reply chains could be visualized. Centrality measures are claimed less useful when only an individual account is analyzed since centrality measures would be greatly biased towards the owner of the account. Strength of ties is used instead. It is a measure for the amount of communication between nodes representing email unique addresses. Both the line width and the blue node size is determined by message count. See Figure 19:

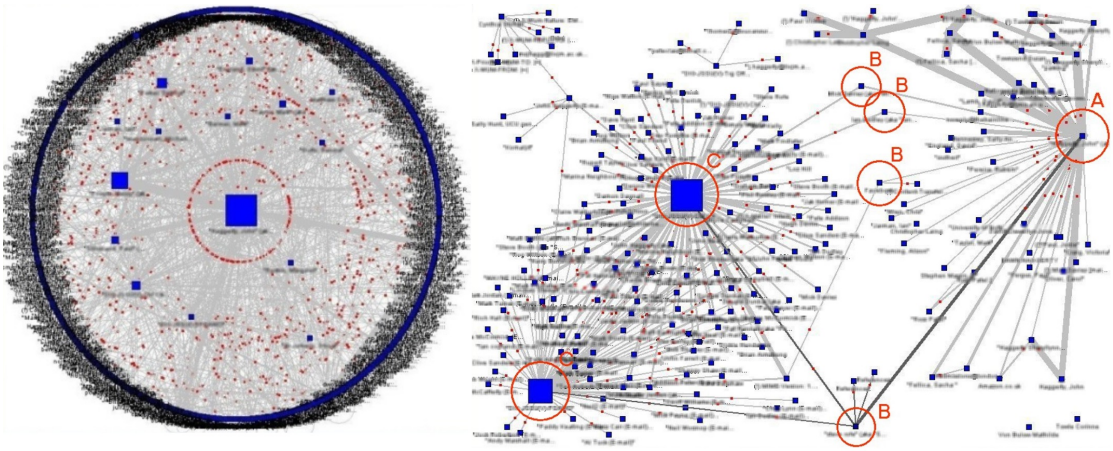


Figure 19: Left: All the 976 email messages with 646 address edges. The mailbox owner as the biggest one, centered in the middle.
Right: One of the folders inside the email account is selected and 4 nodes are circled and marked with a B. These are the *bridges* between networks. Illustration from [68]

Commercial 1: Nuix Forensic Desktop

Nuix Investigator[17] is an email analysis tool. It supports a variety of email storage formats. It performs language detection which is used for filtering messages. It can visualize graphs in a similar fashion to [16] (previous one) and has filtering options to set a minimum message exchange number for inclusion. It searches both text and attachments. The program is Windows based and has the look and feel of Microsoft Office applications.

Commercial 2: i2 analyst notebook

IBM offers a tool called i2 analyst notebook[18] designed for tracking entities movements and communication. A case scenario [69] explains how an investigation into a terrorist network can unfold. Methods are visualization of visited locations on a map. Spotting frequent calls on a time of day heat map and correlation searches across data sources to find matches close in time and/or location. No information on native email support was found. See Figure 20.

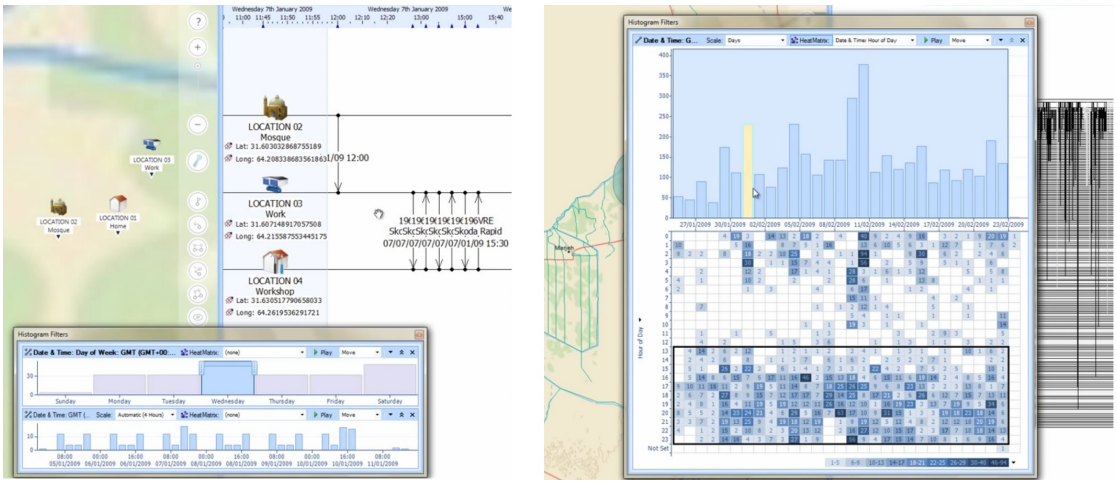


Figure 20: Left: Movements between locations, filtered on Thursdays. Locations are shown on the map. Right: Phone call logs as timeline histogram and time of day heat map. Selecting days and hours filter highlights the corresponding calls in the raw data view behind. Illustrations from [69].

Example 5: Visualizing filtered subsets

Key word searches are commonly used for filtering relevant messages, but is steadily getting slowed down because of the increasing amount of documents to be scanned and indexed [67]. Filtering on date ranges and duplicate removal are explained as important preprocessing stages. Another problem is how to deal with users with multiple accounts, called alias resolving. Experiments on the Enron dataset by the author found 252,956 unique messages of the total 517,431 messages. Determination of the users addresses was based on the *From* field in *Sent items* folders.

[17] `http://aos.com/en-us/products/nuix_forensic_desktop/index.html`
[18] `http://www-03.ibm.com/software/products/no/analysts-notebook`

A *pixel based* timeline is used in a compact representation of the combined messages count sent and received each day for the 150 accounts. It is used for visualizing filtered results of search queries as shown in Figure 21:

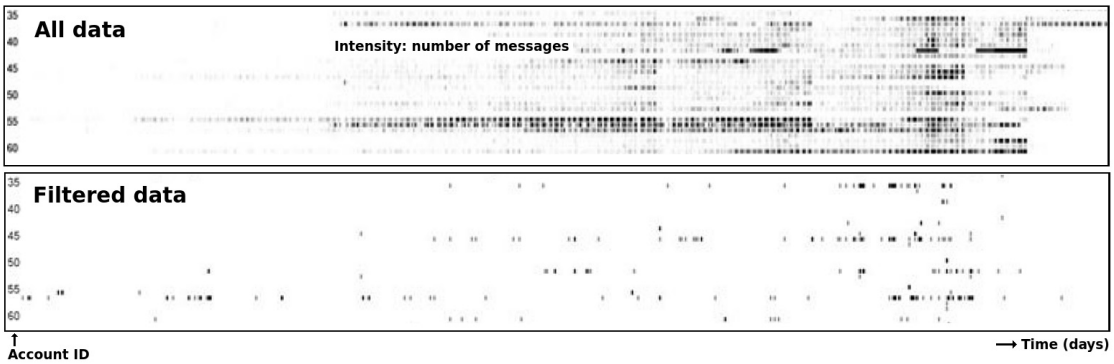


Figure 21: Pixel based compact representation of account activity. Active accounts on a search term can be spotted when filtering before rendering. Illustration from [67], modified with descriptions.

Commercial 3: Relativity

Relativity[19] is a commercial general purpose indexing tool for handling of huge amounts of various documents such as PDF's, office documents and email. It is web based and it indexes the imported content for fast search. It has collaboration support such as marking document relevancy and it takes care of access control to documents.

[19] `http://kcura.com/relativity/`

2.6 Testing methodologies

Testing or benchmarking is done in order to measure the effect of something compared to something else. Examples such as how accurate a category can be classified, how fast an algorithm can be speed up or what the general opinion on a topic in the public is. Testing of how good visualization techniques can aid perception is a very hard task because there is no golden standard to compare against.

The authors of [70] performed a study where 3 experts were tested in a synthetic fraud case where fraudulent web based mail and Internet browser history were mixed in. A modified version of Autopsy[20] was used to record what processes the forensic experts usually performed. The goal was to measure what functions were used the most measured in click patterns. No formal results were published but a few lessons learned was summarized: It is difficulty to recruit experts and testing is very time consuming and rigorous. It was also found that using a tool the experts were not familiar with is distracting. Many uncontrolled factors such as knowledge of the tool, experience of solving a particular problem and personal preferences can interfere.

The authors of the Cyber Forensic TimeLab tool [51] performed a test of their solution compared to the commercial Forensics Tool Kit[21]. The goal was to determine the usefulness of timeline functionality at the time not available in FTK. 12 participants were divided in two teams, one per tool. A credit card fraud scenario was created and performance was measured by time usage and how many correct answers were found. Questions consisted of identifying the stolen credit card number, which bank account money was transferred to and other involved identities. The participants had in general good computer skills, but "little to some" knowledge of digital forensics. Despite the background, accuracy was almost similar across the groups, but the timeline functionality improved the speed of answering considerable. This approach can be biased based on the choice of questions and choices made during development of the experimental case.

The authors of Change-link [55] asked co-workers with experience with tool development and verification to rate their program after using it for a while. Rating was based on a scale from 1 to 6. Questions such as how efficient certain visualizations where and how easy it was to navigate both achieved good scores. Questions related understanding of what to look for scored lower.

A 4th alternative is to perform a qualitative case analysis where different sets of test data is used to highlight situations where a certain method performs well. Examples of this approach are found in [16] and [65]. The problem with this approach is the way it is often used. It is simple to find a situation where a visualization is useful, but there might be a majority of examples that are less clear and situations where no benefit can be found. It is equally important to show situations where the method does not work.

[20] Autopsy: Graphical User Interface for The SleuthKit `http://www.sleuthkit.org/`
[21] FTK at `http://www.accessdata.com/products/digital-forensics/ftk`

2.7 Enron

Enron was a global energy company with thousands of workers founded in 1985. It went bankrupt together with an audit company [71] back in 2001 after the revaluation of fraud. Email messages gathered during the investigation were originally released to the public by the U.S. federal energy regulatory commission (FERC) after their investigation [72]. The case ended in 2006.Key leadership figures were convicted of hiding billions of dollars in debt. The president and chief operating officer Jeffery Skilling, Kenneth Lay (CEO), Andrew Fastow (chief financial officer) together with Ken Rice were (among others) found guilty[68].

The email accounts in the dataset are mostly from individuals in senior management positions and contained originally 158 individual users [73]. Multiple problems with the dataset were discovered and it was sanitized by William Cohen. He reduced the number of accounts to 150. It is known as the *CALO version*. Others have later found that Stephanie Panus had two account folders The same for Lawrence G. Whalley. The number of users is down to 148.

The CALO version with 150 user accounts is used in this thesis. Many preprocessing steps has already been run on this dataset [74]:

- Merging of accounts.
- Messages have been redacted.
- Invalid email addresses have been replaced or set to *no_address@enron.com*.
- New message ID's were created.
- Dates have been converted to a standard timezone.

The dataset has duplicate messages. 252,956 unique was found in [67].

Tread merging has been performed in [75]. The authors used a combination of words in title for messages between the same users found and found 101,786 threads. 30,091 with more than 2 messages.

3 Method design & implementation

This chapter documents the design of a modular web based framework for aiding research in email. Preprocessing is needed in order to visualize and interact with a set of email accounts, and is complicated by the variety of storage formats. The first subchapter explains important email header details followed by a numeration of questions investigators might want to have answered from a set of email accounts. It is followed by a subchapter on the design choices made and ends with important implementation details of the developed prototype.

3.1 The email format and guiding questions

The first step in any data analysis is to look at the raw data in terms of what structured and unstructured information it consists of. What data can be extracted and what information can be deduced. Email messages were initially standardized in RFC[1] 822 [76] (dated 1982). It has since been superseded by RFC 2822 [77] (dated 2001) and recently RFC 5322 (dated 2008).

The email message as defined in RFC 822 had a header part and optionally a body part, both consisting of 7-bit ASCII[2] characters. The body is separated from the header by an empty line. The 7-bit restriction is a consequence of the RFC 821/2821 Simple Mail Transfer Protocol used for sending email to email servers and for forwarding between email servers.

Two examples from the Enron dataset are shown in Figures 22 and 23. Make note of the *content-type* and *content-transfer-encoding* fields. The *quoted-printable* encoding is the reason for the equal signs and the numbers mixed in between the text to the right. Header fields starting with an X- are private non-standardized fields used by email agents.

```
Message-ID: <9996819.1075859083741.JavaMail.evans@thyme>
Date: Thu, 6 Dec 2001 15:12:12 -0800 (PST)
From: danny.mccarty@enron.com
To: rod.hayslett@enron.com
Subject: ENA/Citrus
Cc: stanley.horton@enron.com
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Bcc: stanley.horton@enron.com
X-From: McCarty, Danny </O=ENRON/OU=NA/CN=RECIPIENTS/CN=DMCCARTY>
X-To: Hayslett, Rod </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Rhaysle>
X-cc: Horton, Stanley </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Shorton>
X-bcc:
X-Folder: \Danny_McCarty_Jan2002\McCarty, Danny\Sent Items
X-Origin: McCarty-D
X-FileName: dmccarty (Non-Privileged).pst

Rod,
    I just signed the confidentiality agreement and will track down the
    ENA contracts asap.  I'll let you know when I have them.

Dan
```

Figure 22: The default 7-bit US-ASCII encoding.

```
Message-ID: <23590495.1075854581523.JavaMail.evans@thyme>
Date: Wed, 29 Nov 2000 10:31:00 -0800 (PST)
From: enron.announcements@enron.com
To: all_ena_egm_eim@enron.com
Subject: EnTouch Newsletter
Mime-Version: 1.0
Content-Type: text/plain; charset=ANSI_X3.4-1968
Content-Transfer-Encoding: quoted-printable
X-From: Enron Announcements
X-To: All_ENA_EGM_EIM
X-cc:
X-bcc:
X-Folder: \Eric_Bass_Dec2000\Notes Folders\All documents
X-Origin: Bass-E
X-FileName: ebass.nsf

CONGRATULATIONS!!!
PAT JOHNSON is the winner for =01+The Name This Newsletter Contest=01,.  It=
 was a=20
close call. After receiving over 50 great ideas, the PR Dept had a tough ti=
me=20
narrowing the list to the top ten for Mark Frevert=01,s final decision.  We=
=01,d=20
like to thank everyone who submitted suggestions and a special thanks to Pa=
t=20
Johnson for sending in the winning name EnTouch.=20
```

Figure 23: Quoted-printable using ANSI encoding.

[1] RFC: Request for Comments are peer reviewed publications by the Internet Engineering Task Force describing protocols related to the Internet, originally ARPANET

[2] ASCII is American Standard Code for Information Interchange.

Later evolution of the email standard has added support for multiple body parts such as HTML content and file attachment using the MIME [76] format. MIME is short for *Multipurpose Internet Mail Extensions* and is used for transmitting data that does not fit within the 7-bit ASCII symbol set such as binary attachments and characters in other alphabets. See Figure 24 for an example:

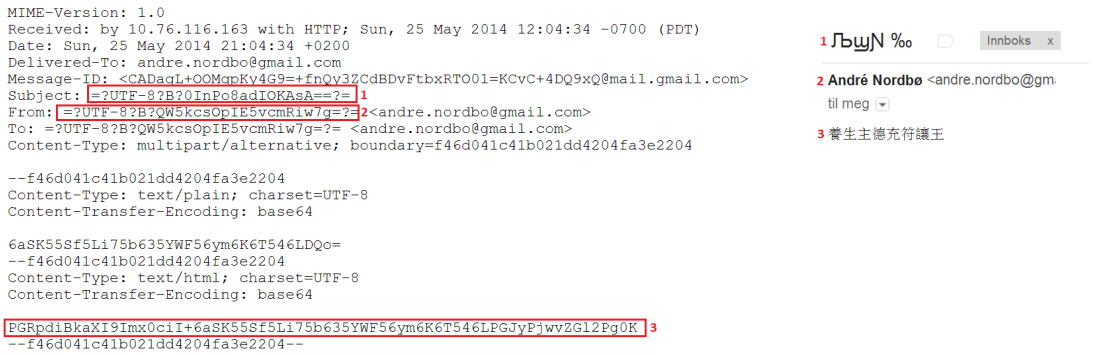


Figure 24: Example of original message from Google Gmail using special characters. Notice the encoding of the body text.

Sent messages will as a minimum have header information about the participants and time of the message. The *Date* field contains the date and time when the message was queued for transmission by the user. The *To* field can have one or multiple email addresses for recipients. The *From* field will usually contain only one address, the one of the sender. It is is not restricted to one by the standard. Optional recipients fields include the carbon copy (*Cc*) and the blind carbon copy (*Bcc*) fields. The *From* field is supposed to be the address of the authenticated identity responsible for sending the message. A *Sender* field can optionally be included if this is not the case, such as when forwarding[76]. RFC 822 defines forwarding as resending a message while keeping the original headers using *Resent-* fields such as *Resent-From*, *Resent-Sender* and *Resent-To*.

The *Subject* is a short optional description of the message content. An email message can also be forwarded in-line, meaning that the original message is put inside the body of a new email message. These forwarded messages usually reflect this fact by prepending the subject line. The same applies for replied messages. Finding interesting ways to automate extraction of knowledge from the written content is an interesting approach, but not the main focus in this thesis.

Numerous questions an investigator would like to have answered can be based on these email fields.

- What are periods of high and low activity? For a user? For a group of users?
- Who works outside of normal working hours? Weekends? Early in the morning or late in the evening?
- Can normal working pattern be determined and thus exceptions located?
- Does a sent message exist as a received message in the corresponding account?
- What messages have been deleted? What are periods of missing activity?
- Who are communicating with the most and fewest unique people?
- Did somebody communicate with a specified person or external company?
- Who talks to most external outside the company. Most internally?
- What external contacts do many have in common and few in common?
- How does the contact surface of a person change over time?
- Who among the persons under investigation has communicated?
- Who communicated during a defined time period?
- Is the communication pattern between peers continuous or spread around?
- Can messages be classified, such as joke, junk message, IT administration, financial related, private etc?
- What is the role of a user in the organization in terms of being an information hub or mainly a receiver? How does it change over time?
- Who are the core individuals in terms of question askers, discussion starters, commentators, experts and administrators. [66]
- How are core individuals connected? Any subgroups? [66]
- Who received a particular attachment? Did many people receive the same attachment?

These questions will guide the design of the interesting visualizations and ways of interacting with email data. All questions are not covered in this thesis. A question could also be formed as a composite of simpler questions, Who sent a message about a certain topic during a time frame to an external person outside the company.

3.2 Method considerations

The goal is to turn raw email data into visualizations that can answer such questions. The design for a modular framework is illustrated in Figure 25. The following 4 sub chapters explain the extraction of messages, parsing of messages, output format and visualization modules.

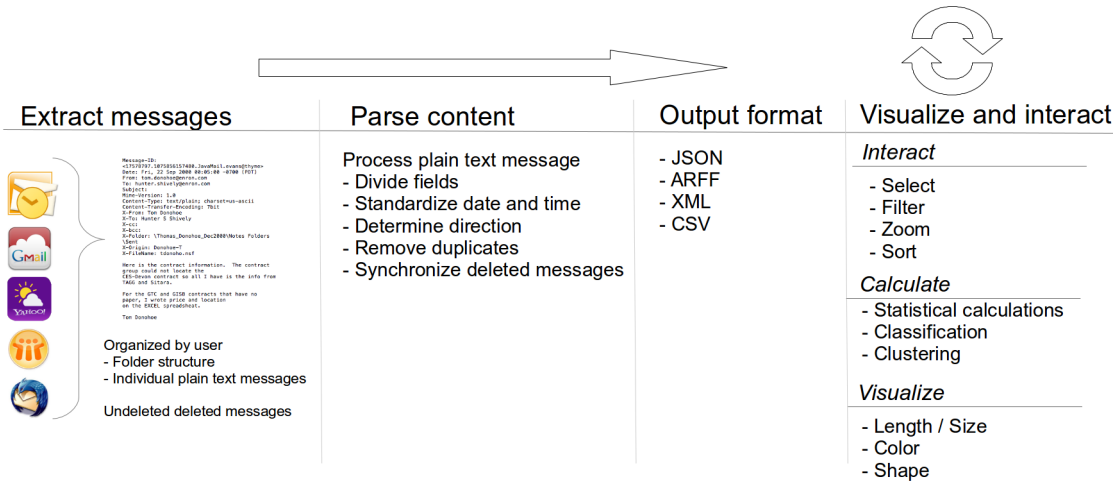


Figure 25: Standardization of individual messages in plain text, parsing of headers and body, extracted data in a standardized format used by a visual system.

3.2.1 Message extraction module

There are two distinct ways email messages can be collected: Proactive and reactive. The proactive way is to have an on-line system collect messages as they arrive. Typically at an email gateway as depicted in Figure 26:

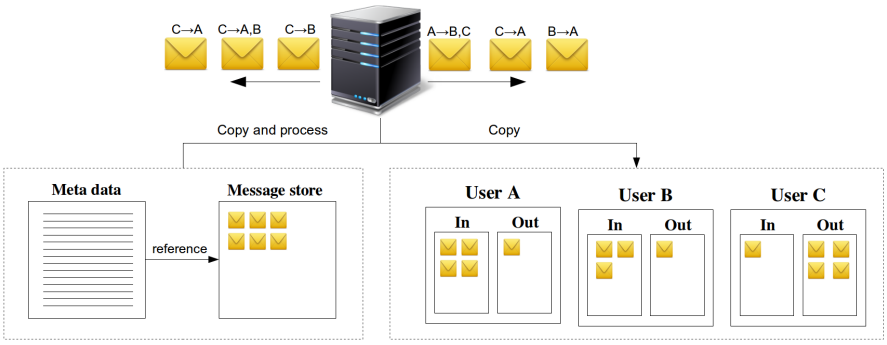


Figure 26: Messages are archived centrally. No messages will be deleted or a duplicate, but the personal sorting is not directly available.

This approach enhances trust in the collected data since collection is separated from the individual users control. Collection can be totally transparent and can potentially be used for

detecting phishing attacks by looking for commonalities among messages sent to a group of accounts. The downsides are privacy and possible legal concerns.

A module at the email gateway can make a copy of every message sent internally and received externally. If a message is internal, one copy could go to the respective *sent* or *received* folder for involved users. Similarly for messages coming in or leaving the organization. Processing and linking can be applied in order to avoid storing multiple copies of the same message and their attachments.

Examples of reactive collection is to get them from users devices, from backups or email servers. This process is typically targeted towards suspects. See Figure 27.

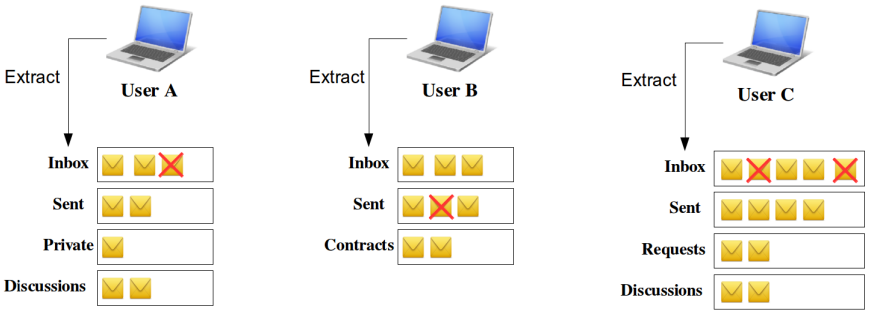


Figure 27: Messages are extracted from individual user accounts. Notice that messages might be deleted. Users tend to organize messages in different folders. Sometimes there will be duplicate messages in multiple folders.

Email applications use different storage formats. This is the main motivation for standardizing the input for the meta data extraction phase. Linux systems traditionally store email in the *maildir* format where messages are stored in plain text as delivered. Each message is stored in a separate file organized in folders. Thunderbird uses a format called *Mbox* where messages are stored in a single file, still in plain text. Microsoft Outlook uses binary encoded files. The version of the Enron dataset used is in the maildir format and is an example of reactive data collection. Storing messages in this format is well suited as a baseline since it is close to the representation when received.

The input of this stage would be any email storage format and the output one folder per account. Inside each folder there will be sub folders replicating the folder organization of the messages in that account. Each message is numbered and stored with full headers and content in RFC 2822 / MIME format. This file format is also known as the *EML* format, although no limitations to RFC 822 only. The primary goal is to preserve the content as close to the original received message as possible.

Recovery of deleted messages is sometimes possible when working reactively, typically in scenarios where the application specific format simply unlinks messages instead of overwriting them. The Outlook *PST* format is an example of such and should be considered when implementing application specific converters.

3.2.2 Parser module (feature extraction)

Each *EML* file consists of a stream of text and must be interpreted in order to be of use. User accounts can be processed in parallel since they are independent of each other, so for each account simply enumerate all the files and parse each one. Do the following operations:

1. Standardize the date time format. A Unix timestamp is recommended.
2. Separate the individual recipients in fields such as *To*, *Cc* and *Bcc*.
3. Decode subject strings and optionally the body text including attachments.
4. Written content can be scanned for keywords or classified.
5. Remove multiple copies of the same message spread across sub folders.
6. Determine the email address of the account. The owner could have multiple email accounts.
7. Determine the direction of each message in terms of being sent or received. This knowledge could be deduced from sub folder names such as *sent* and *inbox*, but is not reliable. A message can be moved between folders. A better strategy is to use the knowledge of the account email address(es).

Messages that has been deleted, but still exist in other users accounts can be reinserted as illustrated in Figure 28. This can be done based solely on the meta data. This idea can be extended to recovery of missing accounts based on acquired accounts. This technique does not guarantee the message was actually received.

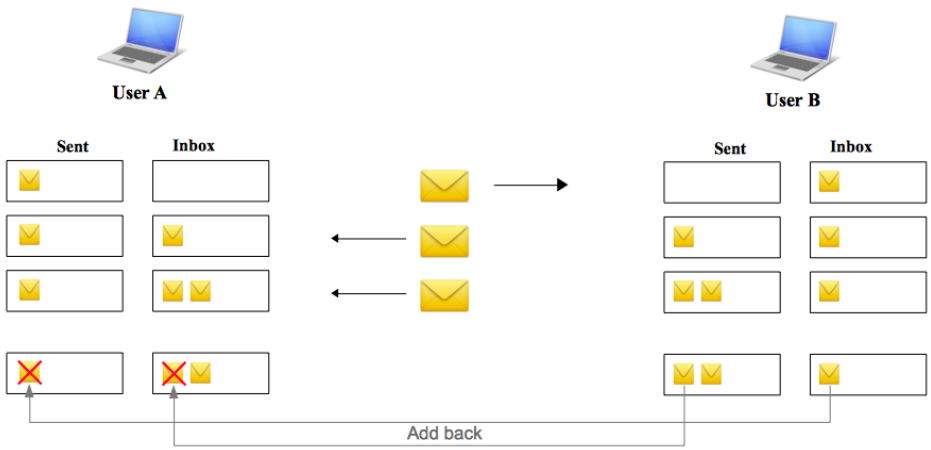


Figure 28: Adding back deleted messages

3.2.3 Output format

A format is needed for storage and exportation of the parsed meta data. Many formats exists and they vary from simple plain text files with comma or tab separated values (CSV and TSV), to more advanced formats such as XML and JSON.

XML stands for Extensible Markup Language and uses nested tags in a way similar to HTML. Data is surrounded by tags in a hierarchal structure.

JSON is JavaScript Object Notation and best practices is to use nested key-value pairs where keys describe data and data is stored as values. This eases object traversal.

ARFF is Attribute-Relation File Format [78] and was developed as a light weight import format for the machine learning program Weka[3]. It defines attribute names and type separately from the data. Data attributes are listed line by line separated by comma. Separating attribute descriptions from the data avoids storing redundant key or tag information.

Other possibilities are using traditional databases or distributed storage systems such as Hadoop[4] and Splunk[5].

Important decision factors are efficiency of the output format in terms of overhead size, how well it integrates with other modules and to some degree how human readable it is.

Web technology is used for supporting the interaction and visualization module of the framework which makes JSON a good choice of data format. It integrates perfectly with JavaScript which is an integral part of dynamic web pages today. JSON has a lot of overhead when keys are used for storing attribute names, so principles from the ARFF format has been used for separating attribute descriptions from the data.

[3]Weka 3: Data mining software in Java, at `http://www.cs.waikato.ac.nz/ml/weka/`
[4]Apache Hadoop at `http://hadoop.apache.org/`
[5]Splunk at `http://www.splunk.com/`

3.2.4 Visualization and interaction

The visualization module is based on Ben Shneiderman's principle:

> "Overview first, zoom and filter, then details-on-demand" [32]

Based on his classification of data type taxonomy it is clear that *network data* (email addresses) and *temporal data* (time) are the key data types for email meta data.

3 primary views have been defined for covering the path from overview to details: A high level *dataset overview*, individual *account detail views* and a *message content view*. Web technology is used for interlinking the views and thus support zooming in on the data down to the original headers and message text. Hyper-linking also allows for relating between accounts and time periods.

Benefits of using web technology is primarily modularization. Workload can be split between server and client, and the client is not restricted to a particular operating system. The server can track progress of read messages and share this information with a team of investigators. Messages of interest can be marked and added to a dedicated case timeline with comments in order to aid comprehension of occurred events.

Dataset overview

This view concentrates on comparing activity among users to get an overview. Common periods of activity, aggregates of hourly and weekday statistics, external and internal communication are examples. The main idea is to use *small multiples* to compare interesting statistics among the users. Examples are message distribution in terms of messages over time, per weekday and hourly as illustrated in Figure 29:

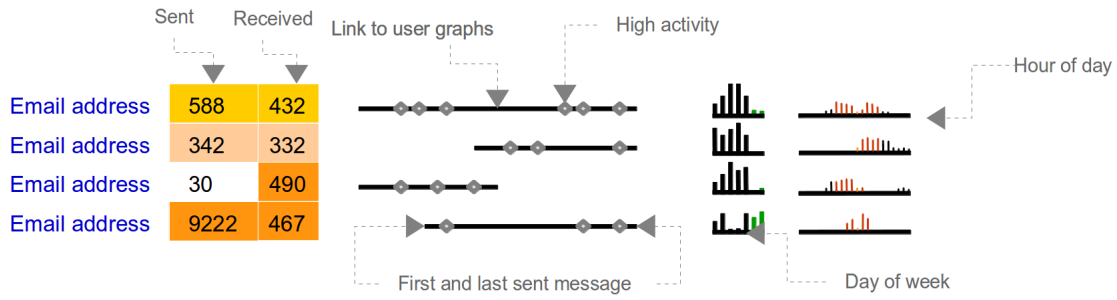


Figure 29: Clicking on fields such as day of week or hour of day should take the investigator to more detailed visualizations.

This design helps the investigator get a sense of periods with high and low activity and compare message statistics like internal and external communication at a glance. The second principle is to normalize values as percentages of the users' message count. Background color intensity can be used for easy comparison of numbers across users similar to the heat map technique used in [62]. Statistics are mainly based on sent messages since received messages to a lesser degree tell a story of the users activity.

Account detail views

Several detailed views can be created. Activity level of a user can be measured in number of messages in each direction over time. Aggregating daily, stacked weekly is an interesting way to see how messages are distributed over the week. Periods of apparently missing activity can be spotted. Saturday and Sunday can be colored differently so to make them stand out. The number of unique communication peers over a time period is another interesting measure for visualizing how the contact surface of a person changes over time. See Figure 30 for illustrations.

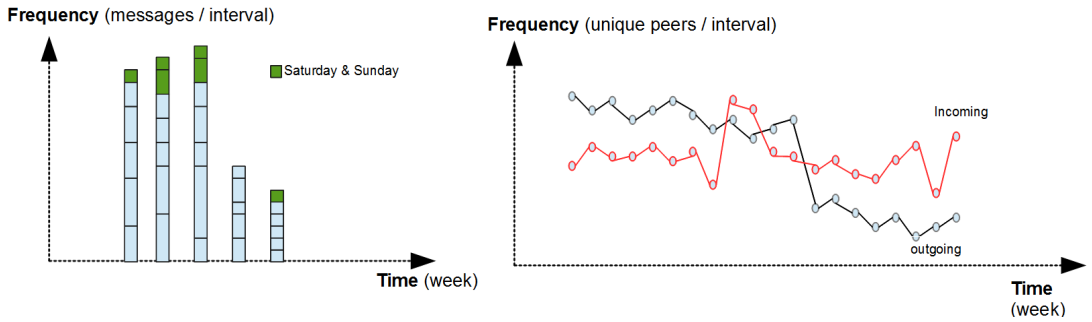


Figure 30: Left: Design for user activity in weekends. Right: Design for tracking contact surface over time. Clicking on a day takes the investigator to the messages for the clicked days.

A time of day plot [59] can give an understanding of when a user was working early and late, and to spot changes in working patterns. This is illustrated in Figure 31:

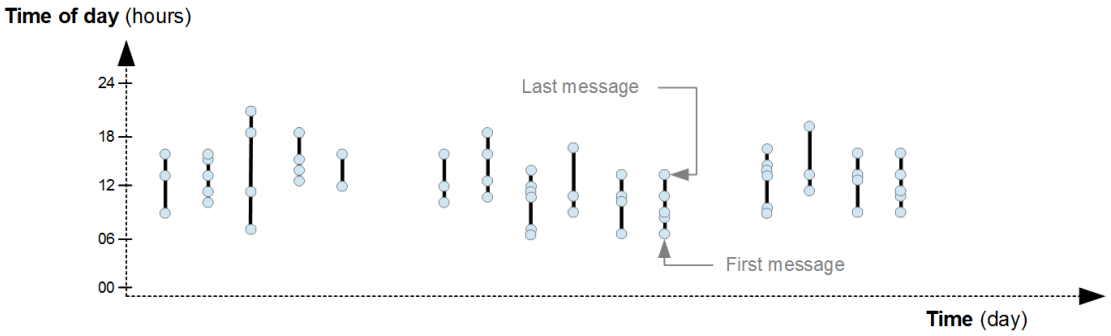


Figure 31: Design of a time of day plot

A communication breakdown can be created by grouping all messages based on email address. It can be done for both directions and plotted over time. It can answer questions related to communication patterns between peers. Is the communication spread evenly or is it interval based. An example of design is illustrated in Figure 32.

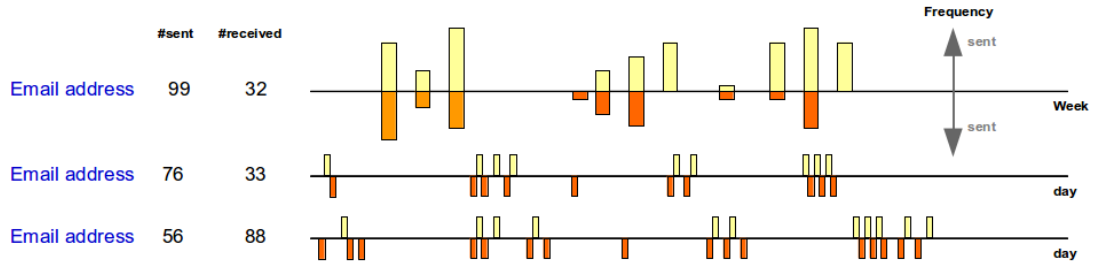


Figure 32: Number of messages sent and received are split and mirrored so that sent messages are growing up, while received messages are growing downwards. Various aggregation levels could be used. Clicking on a histogram takes the investigator to all the messages between the chosen addresses.

Message content view

This is where the interaction to the actual content takes place. The extracted meta data for all messages of an account is sorted by time and displayed in a table. Each message is represented with date and time, direction, subject, size and other relevant meta data. When a message is clicked it is loaded via asynchronous requests and displayed, inspired by [51]. The heat map technique can be used on numeric meta data for easy comprehension of the number as compared to other numbers of the same meta data type. A simple concept illustration is shown in Figure 33.

Read messages are marked and tracked by the server. What makes this interesting is interlinking between accounts. Clicking on an address of a received message takes the investigator to that particular time in the other account. Another interesting approach is to let the investigators select important messages and add them to a case specific timeline.

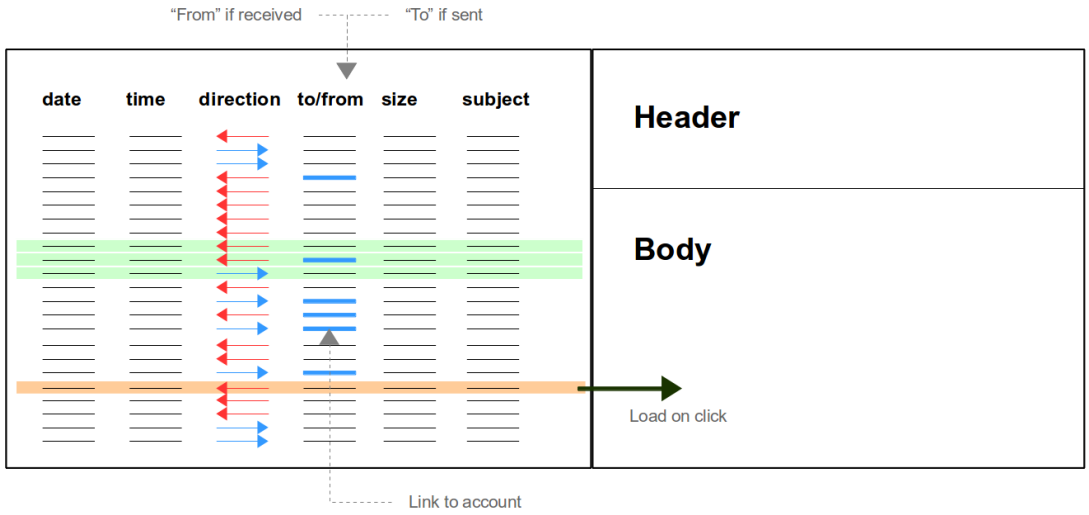


Figure 33: Design of a message content view

3.2.5 Filtering options

Overview, zoom and details has been covered and what remains is filtering. This is a list of interesting ways to filter data:

- Filter out everything except for the accounts under investigation.
- Filter in or out email sent to many recipients.
- Filter in or out short chat sessions of peer to peer communication.
- Filter messages on specific days of week or times of day.
- Filter on word in topic or message body.
- Only show unread messages from the forensic experts perspective.
- Filter in or out messages to external recipients outside of the corporate domain.
- Filter in or out recovered messages.

3.3 Implementation

Implementation has been the means for experimenting with ways to interact and visualize email data, and is not a goal in by itself. Detailed explanations of the code base is therefore put in Appendix D and source code is listed in Appendix E. A brief overview is given bellow.

The process of implementation consist of many decisions. What programing language to use, available libraries and internal representation of data. *Python* (version 2.7.5) was chosen as programing language for portability and transparency reasons. It can run on most platforms and the code is translated to machine code at runtime. This makes Python code slow as compared to precompiled code, but it makes it very easy to inspect the code and make modifications. This is especially important in forensics where it is key to be able to understand and verify the algorithms used. Python programs are typically run in command line terminals. An attractive alternative is *iPython notebooks*[6] where code is entered in a web interface, sent to a python kernel and results of execution returned and displayed in the browser. Python libraries used are partly based on inspiration from a talk by Ron Bodkin on big data analytics [79].

Django[7] (version 1.6.2) was used for server side functionality. Django is a web framework for Python. It automates many of the common background tasks performed by typical web sites such as database handling, session management and user accounts. It also separates code from presentation by use of a template engine. JavaScript is used for client side visualization and processing.

[6] The IPython Notebook at `http://ipython.org/notebook.html`

[7] Django: The web framework for perfectionists with deadlines at `https://www.djangoproject.com`

An overview of code modules and how they are chained together is shown in Figure 34:

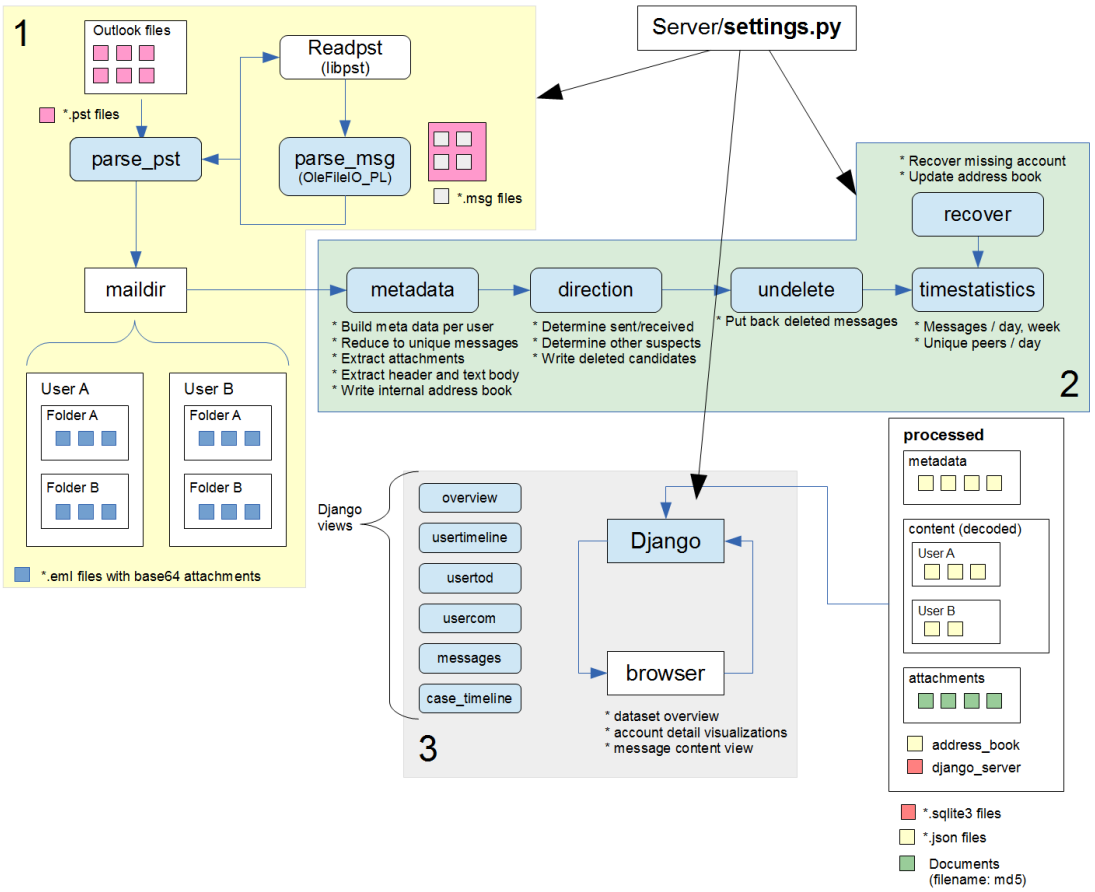


Figure 34: Implemented modules and how they are chained together.
1 yellow: Deals with the conversion from Outlook *PST* files to a plain text *maildir* format.
2 green: Deals with meta data extraction. The *recover* program can optionally be run after the other sub-processes has completed.
3 gray: Is the visualization server module.

All shared settings are stored in the *settings.py* file. The variable *root_base* specifies the full path to the root directory where the current project is hosted. This folder should have a folder named *pst* with *PST* files for every account under investigation. Everything else is build automatically. Optionally import messages in *maildir* format directly in a folder named *maildir* and skip parsing of *PST* files.

An SQLite3 database is created alongside the extracted output to hold status of what messages has been read and added to the case timeline. A small script called *runall.py* takes care of running all the code modules in order and it also starts the development server once done. The following 3 subchapters explain these modules.

3.3.1 Email extraction: Converting PST files

A method for extracting messages from Outlook was needed based on feedback from Økokrim [9]. Most large organizations have standardized on Microsoft Exchange technology and many home users also use Microsoft products for email. Microsoft Outlook stores email data in either PST or OST files [80] locally. OST files are intended for off-line work towards an Exchange server. It is often reasonable to gather the account data directly at the Exchange server [81]. The Exchange server might be configured to keep deleted messages, and additional information available in the Exchange logs [9][82] enhance the argument for centralized collection.

A *PST* file can be thought of as a file system containing many individual *MSG* files representing messages. The *readpst* program as part of libPST [83] can be used for converting MSG files to maildir format, as mentioned in [16].

Comparisons between original headers as shown in Gmail and the output of *readpst* showed that many message headers were missing in the readpst output. Dates had been transformed to GMT timezone stripping away time zone information. *From* and *To* fields usually contain a description and an email address, but often only the description was left. The *readpst* program can also output the original *MSG* files which is the default format of exported individual messages from Microsoft Outlook. This method was chosen in combination with development of a custom program to extract as much of the original message as possible. A comparison between headers from Gmail *original headers* against the readpst output is available in appendix A. The output of the implemented parser is also included.

The *OleFileIO_PL*[84] library was used for parsing the resulting *MSG* files. OleFileIO_PL supports both pre- and post 2003 versions of the *MSG* format. Microsoft has some documentation on the format [85] and a summary of what identifiers are needed was found in [86]. One major issue when dealing with arbitrary text strings is the determination and correct encoding of various character sets. The fall back is set to *iso-8859-1*

Detailed descriptions of inner workings are found in Appendix D.1

3.3.2 Meta data extraction: Parsing EML files

The output of the previous module is a folder called *maildir* containing individual email messages categorized in folders per user account. Four programs are run in sequence in order to extract the meta data and perform preprocessing of it. They are *metadata.py*, *direction.py*, *undelete.py* and *timestatistics.py*. A fifth program *recover.py* can then be run manually in order to recover an account based on the preexisting accounts.

The programs utilize the Python multiprocessing functionality in order to run jobs in parallel. One meta data file is created per account, and each consecutive step will read the meta data file from the previous step, add more meta data to it and save back to the file. The structure in these files have attributes and data separated in order to reduce the size of the files, even though this is against best practices of using key-value pairs.

Email addresses are stored in two variables: The sender and receivers. The first is a string and the second a list of strings. All receiver fields such as *To*, *Cc* and *Bcc* are combined. Determination of account address is based on the folder name of the account, the configured domain and the most frequent addresses seen in the *From* field. The rationale is that *From* fields of messages originating the current user will have the same address, while the *From* fields of incoming messages will have several addresses and likely a lower frequency. This is not guaranteed though. See Figure 35 for an example of address distribution of an Enron employee:

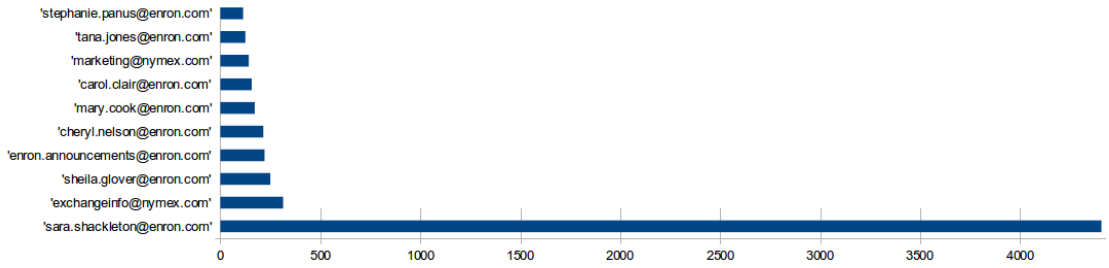


Figure 35: This is the top 10 addresses in the *email_sender* field of the user *shackleton-s*.

A scenario not covered by this implementation is when a user has more than one email address.

Direction of a message is determined based on the knowledge of the owners email address. Recovery of deleted messages from other user accounts is done using the logic shown in Figure 36.

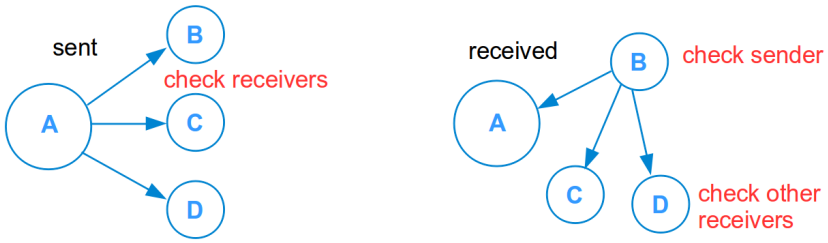


Figure 36: Only the receiver list is relevant when the message was sent. Both the sender and other receivers are relevant when the message was received.

Figure 37 summarizes and gives an example of the resulting JSON meta data format. Detailed descriptions of inner workings are found in Appendix D.2.

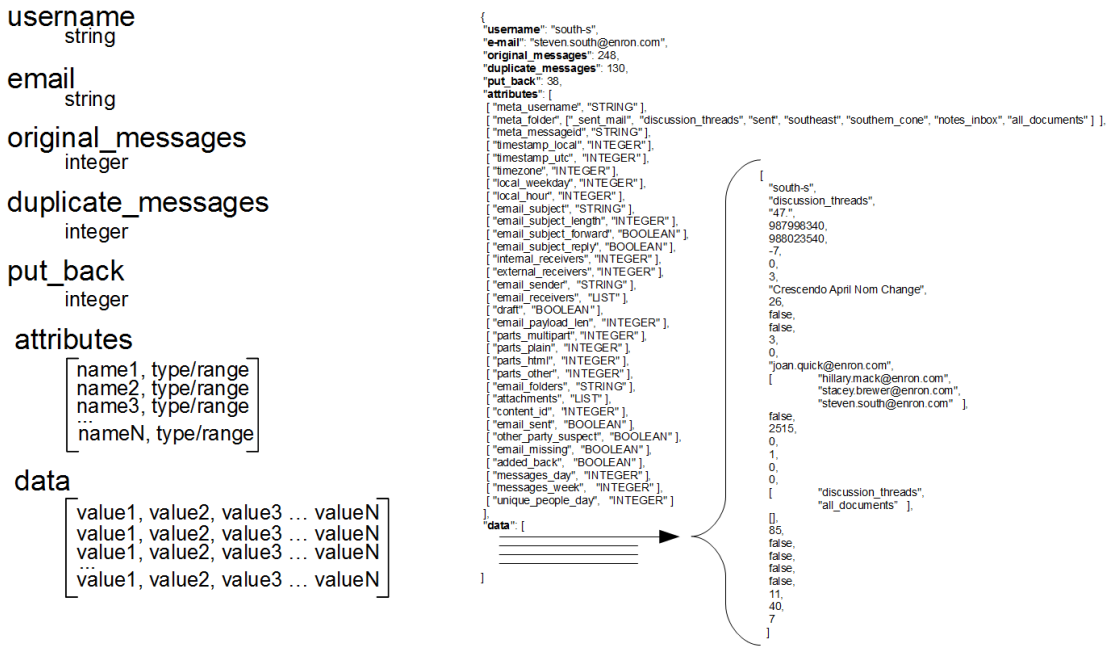


Figure 37: Left: The structure of the meta data JSON file. Right: An example using real data from the Enron dataset.

3.3.3 Visualizing using Django and JavaScript

This module takes as input the JSON metadata files. Detailed descriptions of inner workings are found in Appendix D.3. Description of visual output is given in the following sections:

Dataset overview: Description

The *dataset overview* is the welcome screen presented to the investigator. It shows a table of users filled with graphs and various statistics. Red color is used to present values independent of direction, while green is sent and blue is received. See Figure 38:

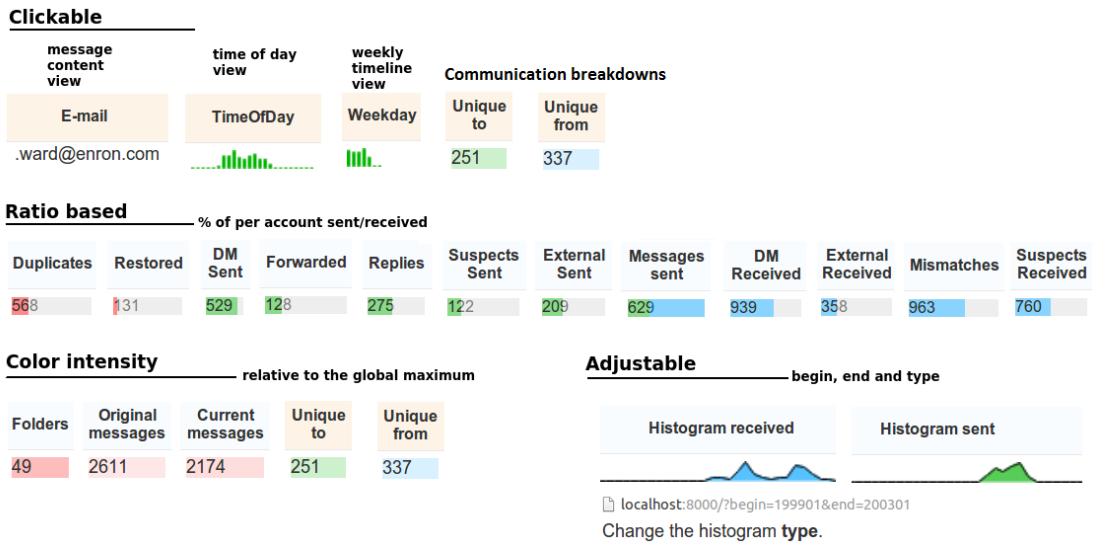


Figure 38:

Row cells with light yellow background link to other visualizations or directly to the list of messages. A percentage based visualization is used when the number is compared to the users' total sent or received message count. Color intensity is used for comparing values globally. The *folders* measure is one such example. A user with a relatively high number of folders compared to other users get a high contrast. Histograms can be controlled by the *type* button between *binary* and *histogram mode*. A time windows can also be set.

Message content: Description

The *message content* page displays meta data for a selected account. See Figure 39:

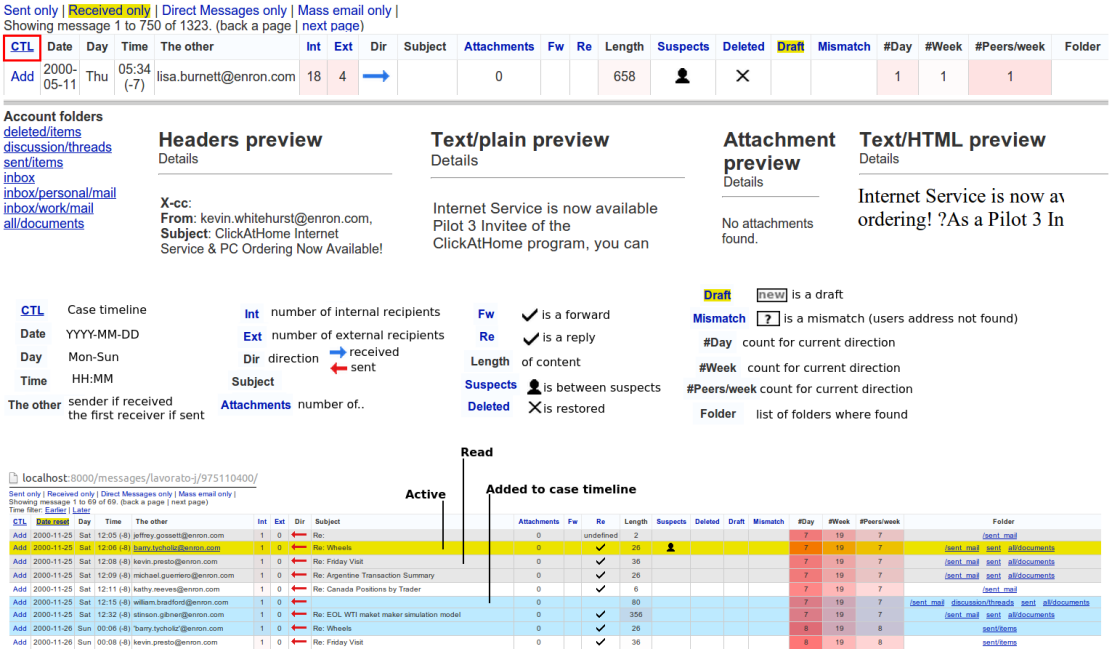


Figure 39: General filters and pagination at top followed by messages in chronological order independent of folder. Bottom half contains preview area for message headers and content preview, loaded dynamically. Notice the button for reaching the *Case timeline marked with a red square.*

The top row of filters allow showing only sent and received messages, messages with only 1 recipient and messages with many recipients (more than 10). A yellow background is used when the filters are active. Many of the table header names are also clickable filters such as *Suspects*, *Deleted*, *Draft* and *Mismatch*. A maximum of 750 messages are showed simultaneously. Pagination is used for moving back and forth. Filtering is performed before displaying and will determine the maximum number shown. A weekly back and forth navigator pane is shown if *Date* filtering is enabled.

Recognized email addresses from an address book are replaced with links to that users messages. The *Date* timestamp filter is set to the same period as the current message the address belongs to. This allows for quick navigation between accounts. Folder names can also be clicked for filtering only the selected folder. Many numerical values are colored by intensity relative to the maximum number for that column.

A button for adding to the *case timeline* is shown in every row. It is called *Add*. Clicking on it creates a new entry in the corresponding database and the background color is set to blue. Clicking on any row will bring up the preview of HTML and plain text together with the original header and links to attachments. Active rows are in yellow while visited are in gray. Clicking any

row triggers an update to another database where all visited messages are tracked. This database is loaded every time the page is requested from the server so that everyone can see what has been read.

Case timeline: Description

The *case timeline* shows messages manually added from the *message content* page. Clicking on an element takes the investigator back to the account and time where the message was added from.

Various messages were selected at random and added. Figure 40 show how it might look like when populated. The plotting library has great support for zooming where elements are dynamically unstacked when more room is available.

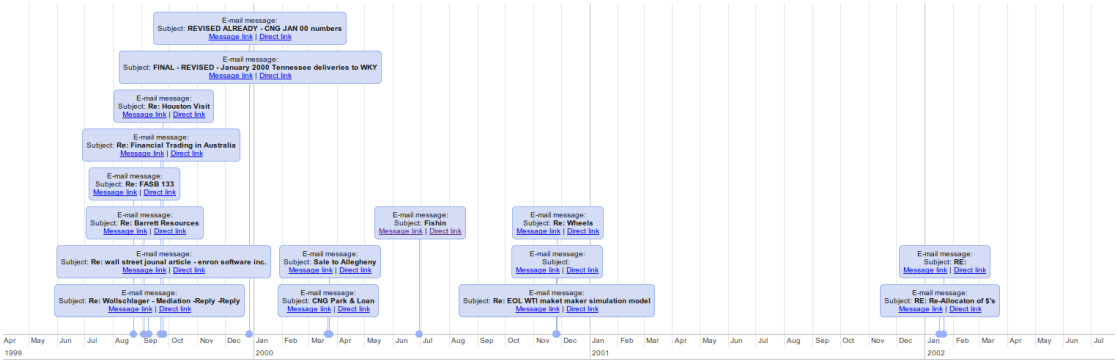


Figure 40: Example of populated case timeline. Each blue box is a message with subject line and links to the corresponding email account.

4 Experiments, results and discussions

The methodology as described in the previous chapter consisted of design and implementation of a modular design for meta data extraction and visualizations using web technology. This chapter documents 3 experiments performed:

1. The first experiment used the preprocessed Enron dataset containing multiple user accounts. The investigative questions defined on page 44 were used for guiding the experiment.
2. Experiment two is performed in order to overcome a major shortcoming of the Enron dataset: It is heavily preprocessed and it is more than 10 years old. The author's Gmail account was used as an up to date dataset for verification of the preprocessing steps. The email account was imported via Microsoft Outlook and imported into the prototype in PST format.
3. The third and last experiment explores the possibilities of replacing the web framework with another visualization tool. The commercial tool Tableau Desktop was chosen. Extracted meta data was imported to Tableau and the experiment focused on how to recreate similar visualizations as already implemented in the prototype.

This chapter is concluded with discussions of how visualizations can explain benefits and limitations of automatic outlier detection.

4.1 Experiment 1: Enron

4.1.1 Test setup

The test hardware used comprised of a desktop and a laptop computer. The desktop was equipped with a quad core Intel i5 3570K with 16 GB of random access memory running Ubuntu 13.10. The laptop was an Apple Air equipped with a dual core (4 threads) Intel i5-3427U with 4 GB of random access memory running OS X 10.9.2. Both are equipped with solid state drives. Google Chrome was used as the default web browser. The dependencies are *Python*, *Django* and *Pandas*.

The CALO version of the Enron dataset was downloaded. The archive [74] is named *enron_mail_20110402.tgz*. It is 443.5 MB compressed and 1355.5 MB uncompressed. The extracted *maildir* folder contains 150 folders with 517.424 files[1]. Early exploration of the containing files led to the discovery that *stokley-c* contained an intermediate folder *chris_stokley*. This sub folder was removed and its contents put directly in the *stokley-c* parent folder.

The maildir was put inside a folder representing the root folder of this investigation project. This root folder was configured as *root_base* in the *settings.py* configuration file. The *company_domain* variable set to "*@enron.com*" and *eml_match_regex* to "*(\d+)*". It will match integers since the files in the maildir folder does not contain file extensions. The file *runall.py* was run with the *parse_pst* line commented out.

4.1.2 Results: Meta data extraction

Initial experiments before implementation of functionality to remove duplicate messages gave these results in terms of the size of the extracted meta data as shown in Table 1.

Size on disk	Content size	Extracted meta data	Extracted data (except To/Cc/Bcc)
2603 [MiB]	1355 [MiB]	**180** [MiB]	85 [MiB]

Table 1: The extracted meta data is approximately 13% of the original information. Notice that about half of the meta data is email address information. The 517.424 individual files has an arithmetic mean of 2,68 KB explaining why the size on disk is much larger than the actual content. Size on disk will depend on the minimum file allocation size.

Determination of email address is based on the information in the account *folder name* and the most frequent *sender address*. 2 pairs of the the 150 account folders have an email collision, confirming the findings reported at enrondata.org [73]. 5 addresses of the remaining 148 accounts were assigned erroneously. Another 5 were based on choosing the most frequent address. This was caused by misspellings in the folder names. This is summarized in Table 2. The data is available in appendix B where the addresses and comments have been appended to the custodian sheet from EnronData [87].

[1] Number of files determined by "*find . -type f | wc -l*"

Folder name	Detected email address	Correct email address	Comment
phanis-s	stephanie.panus@	stephanie.panus@	Duplicate of panus-s. Folder name: us, not is
whalley-l	greg.whalley@	greg.whalley@	Duplicate of whalley-g
dean-c	clint.dean@	craig.dean@	If real name is Craig Dean
hodge-j	t..hodge@	john.hodge@	If real name is John Hodge
hernandez-j	judy.hernandez@	juan.hernandez@	If real name is Juan Hernandez
lay-k	elizabeth.lay@	kenneth.lay@	if real name is Kenneth Lay
merriss-s	pete.davis@	steven.merris@	Obviously wrong. Folder name: extra s
stclair-c	carol.clair@	OK	Folder name: prefix st
gilbertsmith-d	doug.gilbert-smith@	OK	Folder name: missing -
ybarbo-p	paul.y'barbo@	OK	Folder name: missing '
rodrique-r	robin.rodrigue@	OK	Folder name: q instead of g
crandell-s	sean.crandall@	OK	Folder name: e instead of a

Table 2: The email guessing method works for 143 of 148 (150) accounts.

3 accounts have been verified to have multiple alternative addresses and it is likely to be the case for other accounts. One example is the account of *Kevin M. Presto* where the following addresses was observed: *kevin.presto@enron.com*, *kevin.m.presto@enron.com* and *m..presto@enron.com*.

The two duplicate users were merged by putting each of the directories for the two users inside a new folder with the corrected username.

The frequency distributions of the accounts assigned wrong email address is shown in Figure 41. It also contains examples where multiple addresses looks correct.

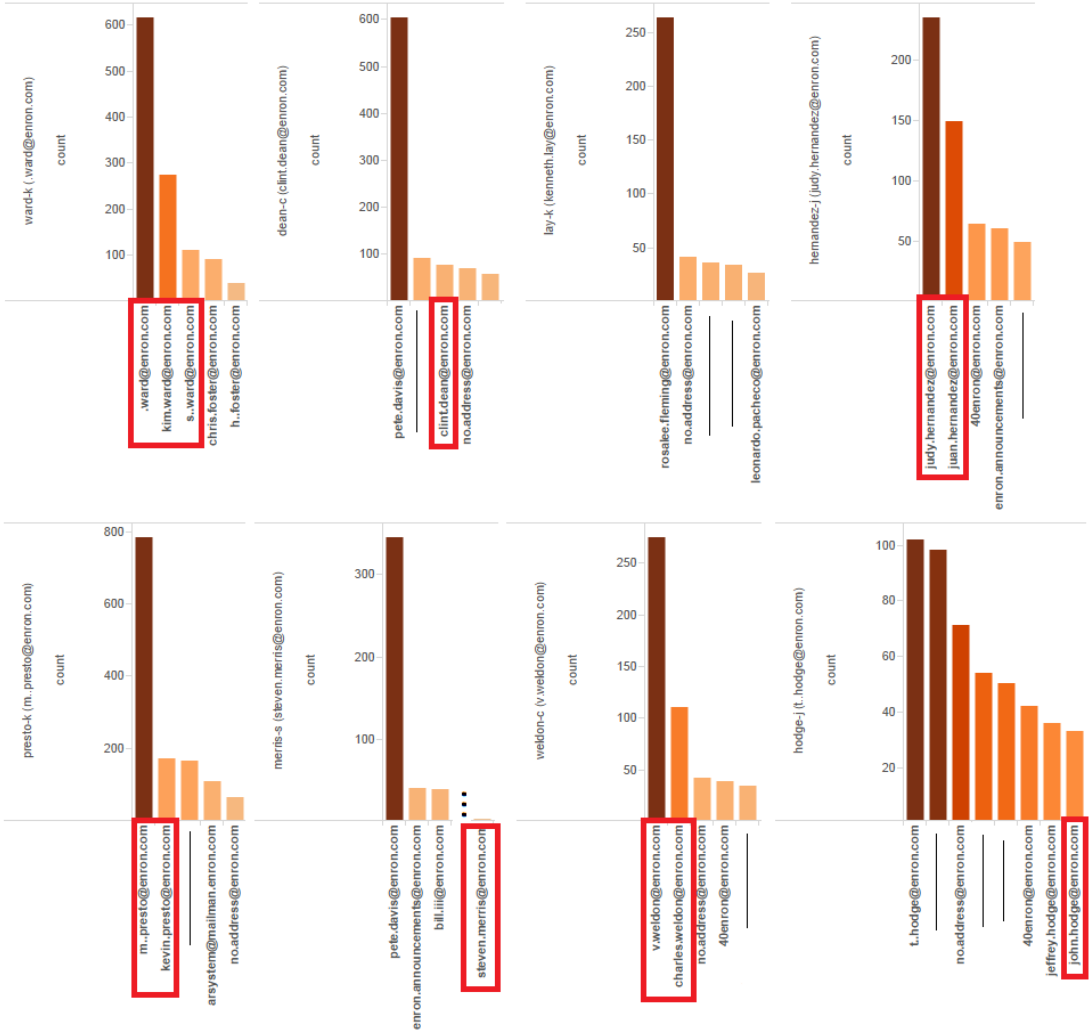


Figure 41: The red markings highlight the correct address(es).

Many explanations can be given for why the users address is not dominating the *From* field of own messages. One factor could be that the user has deleted sent messages. It could be because of a subscription service delivering messages regularly. It could be that multiple email accounts have been merged or synchronized as is the case for the Enron dataset.

Initial experiments revealed that sent messages for accounts in the dataset were missing at the recipient. The updated statistics after implementation of duplicate removal and message recovery are shown in Table 3:

Original messages	Duplicates removed	Messages left	Added back	Total messages
517,424	228,703	288,721	104,482	393,203
100%	44.2%	55.8%	+36.2%	

Table 3: Duplicate removal and undelete: 289k unique compared to the 253k found in [67].

The meta data extraction pass is the most time consuming. Table 4 summarizes the time elapsed on both of the test systems. Both systems run with a number of processes equal to the number of detected CPU cores.

System	Extract	Direction	Add back	Time statistics
Desktop	353 [sec]	5 [sec]	80 [sec]	3 [sec]
Laptop	1038 [sec]	11 [sec]	208 [sec]	24 [sec]
Desktop vs Laptop	2.9x	2.2x	2.6x	8x

Table 4: Note that 517,424 JSON content files are being written to disk in the Extraction phase. Duplicates are unlinked.

Recovery of accounts

It is not unlikely to find interesting email messages sent or received from an account not specifically acquired. 4 individuals were convicted as described on page 41: Jeffery Skilling and Kenneth Lay are both present in the dataset, but Andrew Fastow and Ken Rice are not. Opening the communication breakdown of Jeffery Skilling and a search for *fastow* and *rice* revealed their email addresses. The program *recover.py* was run targeting these two addresses, and the result was two additional meta data files with reassembled messages from all the other 148 user accounts. These meta data files are added to the dataset overview and can be examined just like any other account.

The results for *andrew.fastow@enron.com* returned 372 messages where 8 of them were sent. *ken.rice@enron.com* returned 276 messages of them 7 sent.

A naive verification is done using the *grep* command

grep -rl "andrew.fastow@enron.com" . | wc -l

The result is 1242 files found inside the *maildir* folder.

4.1.3 Results: Visualization and interaction

The Django development server was initiated by the last step in the *runall.py* script. It was accessed typing "*http://localhost:8000*" in the address bar of Google Chrome. The start page is the *dataset overview*. The first apparent observation is the huge histogram gaps as seen in Figure 42.

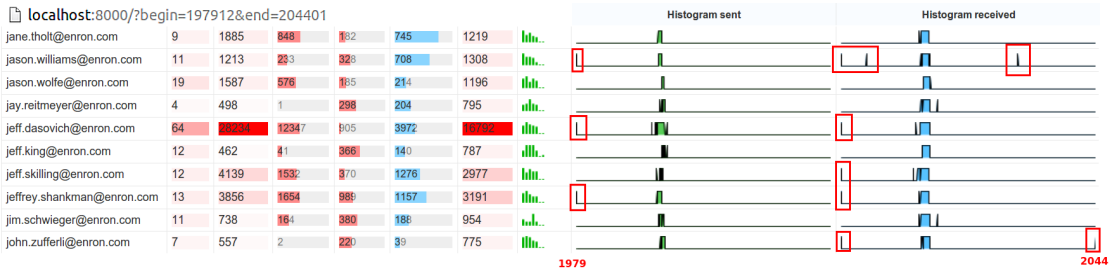


Figure 42: First date observed is December 1979 and the latest January 2044. Notice accounts with strange outlying dates.

The gaps are caused by messages dated 1979 and 2044 in the extremes. The time filter was set to start 199609 (September 1996) and end 200303 (March 2003) and type set to histogram. The whole page can be seen in appendix C. The overview can give hints of accounts communicating mostly internally and which ones communicate a lot to external domains. A high mismatch value might hint of erroneously determines email address.

The first defined investigative question on page 44 is: *what are periods of high and low activity*. All activity sent and received is summarized in the histograms. Figure 43 shows a close up illustration:

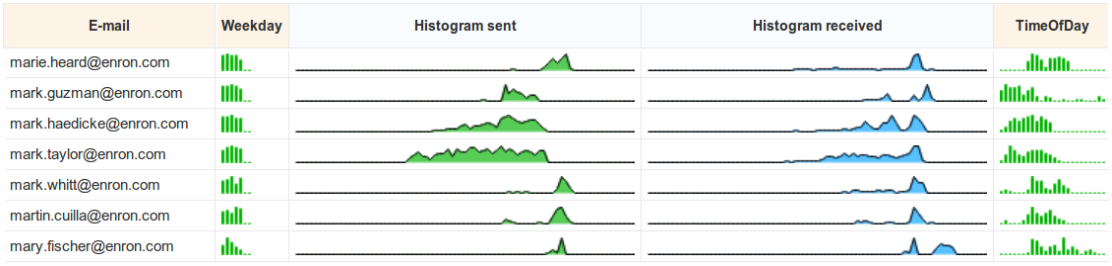


Figure 43: The period of collected data varies greatly among the suspects.

The account with the address *mark.taylor@enron.com* has been randomly selected as an example. The account was active for a long period. His weekday graph shows activity only on Monday to Friday and he is very active between midnight and noon. Clicking on the *weekday* graph brings up the *weekly timeline* as seen in Figure 44:

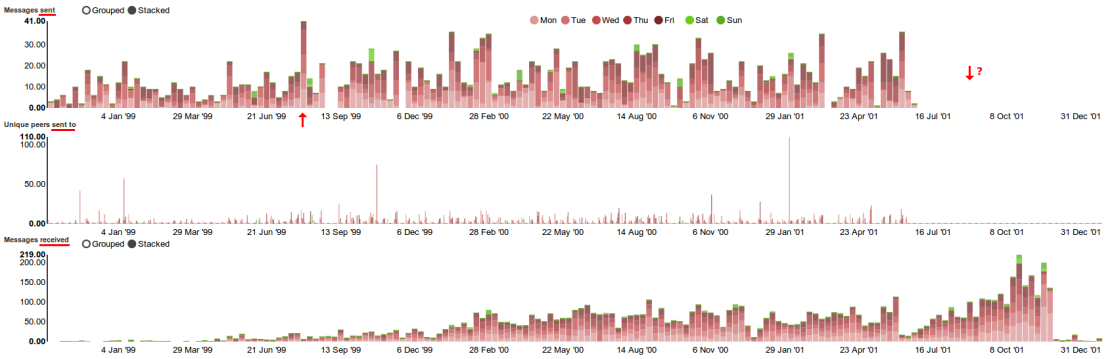
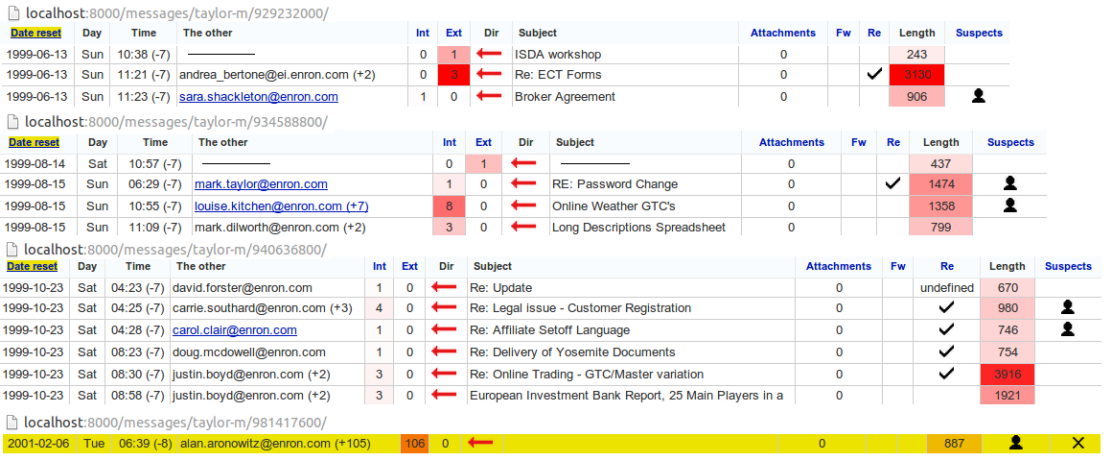


Figure 44: A few periods have a high number of sent messages. No available messages are sent the last half of 2001 while messages are received. Messages to a lot of people have been sent 4 times, the latest midway January 2001.

A few weeks with sent messages in weekends could be spotted in the *weekly timeline*. Clicking these green bars redirects the browser to the *message content view* where the messages can be inspected. A few examples are shown in Figure 45.



localhost:8000/messages/taylor-m/929232000/

Date reset	Day	Time	The other	Int	Ext	Dir	Subject	Attachments	Fw	Re	Length	Suspects
1999-06-13	Sun	10:38 (-7)	————	0	1	←	ISDA workshop	0			243	
1999-06-13	Sun	11:21 (-7)	andrea_bertone@ei.enron.com (+2)	0	3	←	Re: ECT Forms	0		✓	3130	
1999-06-13	Sun	11:23 (-7)	sara.shackleton@enron.com	1	0	←	Broker Agreement	0			906	

localhost:8000/messages/taylor-m/934588800/

Date reset	Day	Time	The other	Int	Ext	Dir	Subject	Attachments	Fw	Re	Length	Suspects
1999-08-14	Sat	10:57 (-7)	————	0	1	←	————	0			437	
1999-08-15	Sun	06:29 (-7)	mark.taylor@enron.com	1	0	←	RE: Password Change	0		✓	1474	
1999-08-15	Sun	10:55 (-7)	louise.kitchen@enron.com (+7)	8	0	←	Online Weather GTC's	0			1358	
1999-08-15	Sun	11:09 (-7)	mark.dilworth@enron.com (+2)	3	0	←	Long Descriptions Spreadsheet	0			799	

localhost:8000/messages/taylor-m/940636800/

Date reset	Day	Time	The other	Int	Ext	Dir	Subject	Attachments	Fw	Re	Length	Suspects
1999-10-23	Sat	04:23 (-7)	david.forster@enron.com	1	0	←	Re: Update	0		undefined	670	
1999-10-23	Sat	04:25 (-7)	carrie.southard@enron.com (+3)	4	0	←	Re: Legal issue - Customer Registration	0		✓	980	
1999-10-23	Sat	04:28 (-7)	carol.clair@enron.com	1	0	←	Re: Affiliate Setoff Language	0		✓	746	
1999-10-23	Sat	08:23 (-7)	doug.mcdowell@enron.com	1	0	←	Re: Delivery of Yosemite Documents	0		✓	754	
1999-10-23	Sat	08:30 (-7)	justin.boyd@enron.com (+2)	3	0	←	Re: Online Trading - GTC/Master variation	0		✓	3916	
1999-10-23	Sat	08:58 (-7)	justin.boyd@enron.com (+2)	3	0	←	European Investment Bank Report, 25 Main Players in a	0			1921	

localhost:8000/messages/taylor-m/981417600/

2001-02-06	Tue	06:39 (-8)	alan.aronowitz@enron.com (+105)	106	0	←		0			887	

Figure 45: Messages sent during weekends. The bottom message is the last peak in the *Unique peers sent to* graph. It was sent to 106 persons and is a request for modifying filling practices for (paper) files since storage space is running out.

Mark Taylor had a strange distribution of sent messages. Only during the night up until noon. The *time of day* view is opened by clicking the *TimeOfDay* graph and shown in Figure 46. Normal working patterns can be determined visually. The pattern is concise up until about June 2001 when the range is lifted up in the normal range of 08-16 hours. This might suggest errors in *Date* parsing. Individual dots also takes the browser to the *message content* view. Many messages has been checked and confirmed to be correct according to the original headers. The error seems to stem from the dataset itself. Another explanation could be time zone related, but it is set to -7 and -8 both before and after the jump.

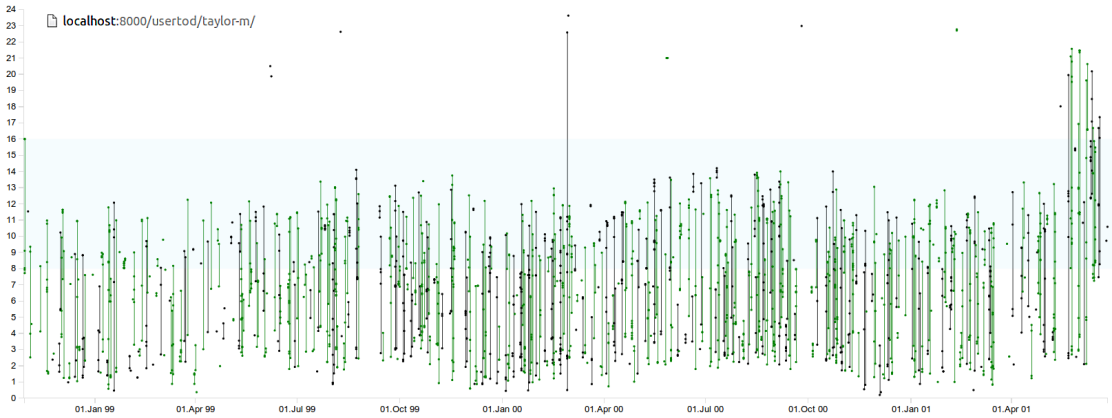


Figure 46: Time of day view: A strange shift in working hours.

The same observation is found for other accounts such as *dorland-c* and *jones-t* in Figure 47.

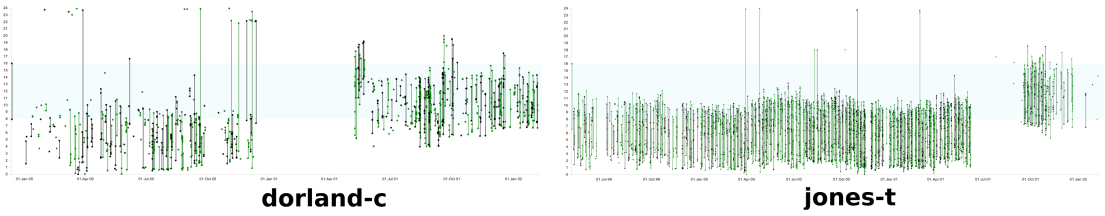
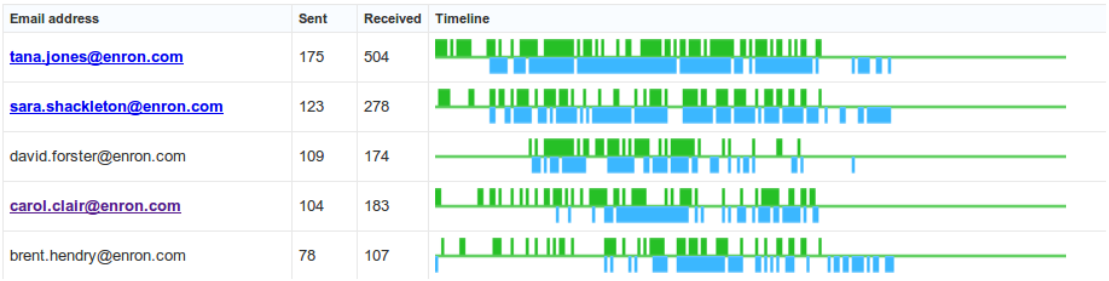


Figure 47: The jump in range is found in multiple accounts around midway 2001

The *communication breakdown* view can answer who are the most active and how messages are spread over time for the individual accounts. The top 5 internal for *mark.taylor@enron.com* are shown in Figure 48:

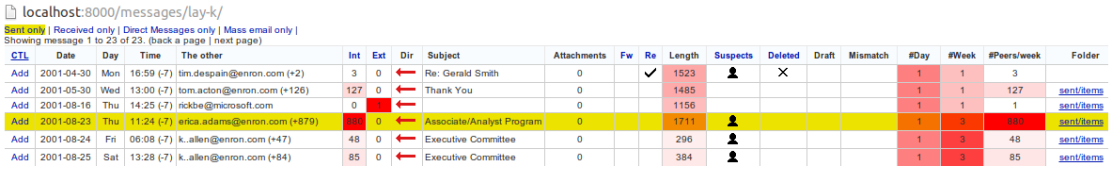


Email address	Sent	Received	Timeline
tana.jones@enron.com	175	504	
sara.shackleton@enron.com	123	278	
david.forster@enron.com	109	174	
carol.clair@enron.com	104	183	
brent.hendry@enron.com	78	107	

Figure 48: Communication breakdown

Communication can be continuous over a long period and be spread in bursts. It can be one way or a two way interaction. The pattern can say something of the relationship.

The *dataset overview* has many other interesting outliers: Kenneth Lay has sent to 1059 unique addresses in only 21 outgoing messages. Exploring this by entering the *detailed message* view with *sent only* filter ON explains this as a message sent to 880 users. See Figure 49. The count of unique addresses is thus a poor estimator which persons communicating with many people.



localhost:8000/messages/lay-k/

Sent only | Received only | Direct Messages only | Mass email only |
Showing message 1 to 23 of 23. (back a page | next page)

CTL	Date	Day	Time	The other	Int	Ext	Dir	Subject	Attachments	Fw	Re	Length	Suspects	Deleted	Draft	Mismatch	#Day	#Week	#Peers/week	Folder
Add	2001-04-30	Mon	16:59 (-7)	tim.despain@enron.com (+2)	3	0	←	Re: Gerald Smith	0		✓	1523		✕			1	1	3	
Add	2001-05-30	Wed	13:00 (-7)	tom.acton@enron.com (+126)	127	0	←	Thank You	0			1485					1	1	127	sent/items
Add	2001-08-16	Thu	14:25 (-7)	rickbe@microsoft.com	0		←		0			1156					1	1	1	sent/items
Add	2001-08-23	Thu	11:24 (-7)	erica.adams@enron.com (+879)	880	0	←	Associate/Analyst Program	0			1711					1	3	880	sent/items
Add	2001-08-24	Fri	06:08 (-7)	k..allen@enron.com (+47)	48	0	←	Executive Committee	0			296					1	3	48	sent/items
Add	2001-08-25	Sat	13:28 (-7)	k..allen@enron.com (+84)	85	0	←	Executive Committee	0			384					1	3	85	sent/items

Figure 49: Detailed message view: 880 recipients

The account with the address *bert.meyers@enron.com* is sending a lot of messages on Saturdays and Sundays and at all times during the day, although mostly in the evening. Only 60 of his 3022 messages are sent. This might suggest this email address is not labeled correctly. His *sent/items* folder has messages from *albert.meyers@enron.com*, not the detected *bert.meyers@enron.com*. Statistics for this user can not be trusted because of this.

Darrell Schoolcraft has forwarded about half of his sent messages. Daren J Farmer's sent replies count for the majority while almost none of the sent messages are forwarded.

john.lavorato@enron.com has a high frequency of messages on Sundays. Figure 50 shows the *weekly timeline*:

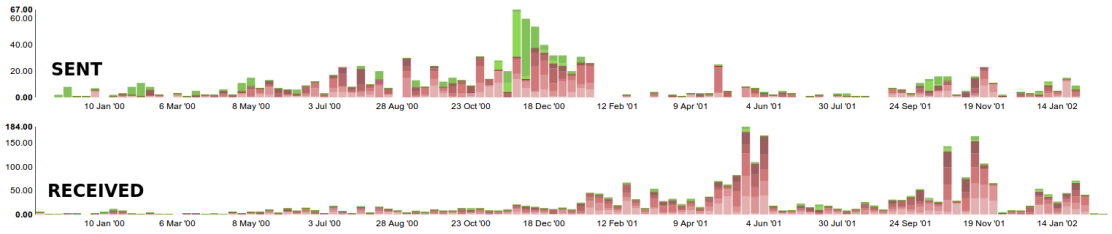


Figure 50: Notice the high peak of weekend activity during November and December 2000.

The answers to what messages has been deleted and communication between acquired accounts can partly be answered using the *message content* view with filters enabled. The *suspects* filter only shows messages where another user from the address book is present. The *deleted* filter is a subset of suspect messages where the message originally did not exist in the current email account. See Figure 51.

localhost:8000/messages/thomas-p/

Sent only | Received only | Direct Messages only | Mass email only |
Showing message 1 to 8 of 8. (back a page | next page)

CTL	Date	Day	Time	The other	Int	Ext	Dir	Subject	Attachments	Fw	Re	Length	Suspects	Deleted	Draft	Mismatch	#Day	#Week	#Peers/week	Folder
Add	2001-06-04	Mon	07:39 (-7)	peter.makkai@enron.com (+3)	4	0	←	CSS Training Program	0			291	👤				1	4	4	sent/items
Add	2001-06-22	Fri	10:05 (-7)	joe.stepenovitch@enron.com (+4)	5	0	←	FW: NYISO Transactions and the Upload	0	✓		738	👤				1	4	5	sent/items
Add	2001-07-17	Tue	14:21 (-7)	f..campbell@enron.com (+4)	5	0	←	FW: NYISO RT Web Application	0	✓		737	👤	✕			1	1	5	
Add	2001-09-21	Fri	10:29 (-7)	kim.melodick@enron.com (+3)	4	0	←	Internal Transition Request	0			435	👤				3	12	6	sent/items
Add	2001-09-24	Mon	05:41 (-7)	peter.makkai@enron.com (+4)	5	0	←	Hourly Meeting	0			134	👤				1	11	5	sent/items
Add	2001-10-08	Mon	09:06 (-7)	dana.davis@enron.com (+5)	6	0	←	FW: Proposed Items of Discussion for Oct 22	0	✓		1753	👤				6	13	9	sent/items
Add	2001-10-12	Fri	10:00 (-7)	jae.black@enron.com (+2)	3	0	←		0			302	👤				3	13	5	sent/items
Add	2001-10-30	Tue	10:16 (-8)	joe.quenet@enron.com	1	0	←	YOUR ENPOWER NUMBER	0			24	👤				1	4	1	sent/items

Figure 51: The username *thomas-p* has sent these messages to other collected accounts.

The *communication breakdown* for the username *sager-e* revealed a rapid pattern of messages from *tana.jones@enron.com*. Clicking this chart shows these messages in the *message detail* view. They are daily reports as seen in Figure 52.

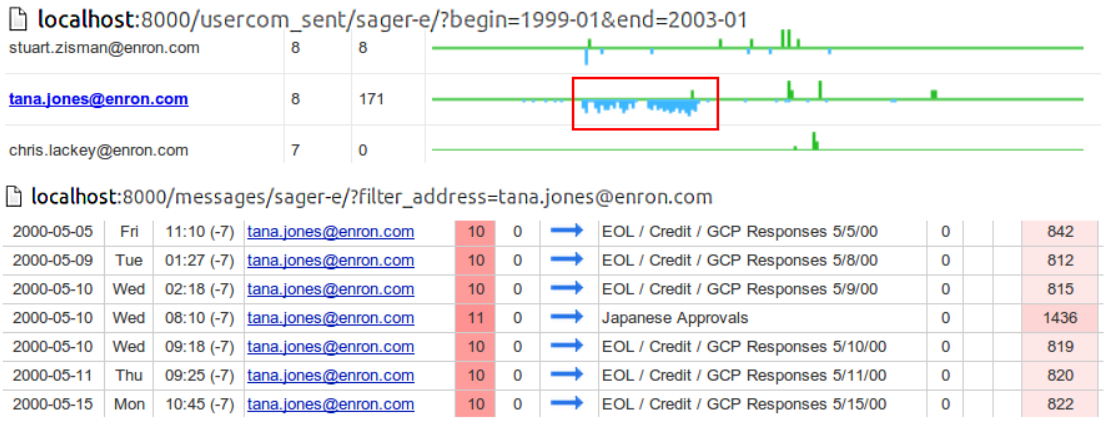


Figure 52: Interlinking allows quick navigation across time and accounts.

4.1.4 Discussions

The *dataset overview* gives a quick overview of periods with high and low activity across multiple accounts. Low or missing activity levels would normally have explanations such as vacation periods or archiving as a consequence of mailbox size restrictions. A gap is still interesting to follow up on. The view also gives a quick impression of distribution across weekdays and time of day, internal and external communication in addition to the number of unique addresses a given person has communicated with. The number of unique addresses in each direction has been found unreliable as some messages sent to large groups can have all the individual recipients split up. A more reasonable measure is the number of unique combinations of recipients. Two messages to A and a message to B, C and D would then count as only two unique combinations.

The *time of day view* is good for visually determination of normal working patterns. Display of received messages would not contribute to the activities of the current user as these messages are not controlled by the user. Outliers such as very early or late could be interesting and the same for messages at regular times of day. Received messages at regular times could be interesting in terms of filtering them. Examples could be machine generated reports.

The *communication breakdown view* visualizes communication between pairs of persons. Addresses topping this view are interesting because they are important to the account owner. It could be close co-workers. The bottom list could also be interesting. Many examples from the Enron dataset suggests it is dominated with addresses resulting from emails sent to groups of people. Filtering out messages sent to a threshold of people could improve the usefulness and limit the amount of unique addresses. A message sent to many addresses is less personal and less likely to contain sensitive information. The communication with various other users can be continuous for a long time period or restricted to short periods. The breakdown per person can help the investigator quickly narrow down who to concentrate on given a time windows of interest.

The mismatch flag in the *message content view* is raised when the users determined email does not match either the sender or recipients field. This does not necessarily imply something is wrong. The message could have been imported, but it is more likely the message was sent to a collective address and delivered using a *Delivered-To* header instead of the traditional *From or To* headers. Adding alternative sender and recipients fields could improve the framework. The major complication with the prototype is the limitation of only one email address per account and that this address is sometimes mistakenly assigned. A mistaken or missing address in the address book have consequences such as messages not marked as *between suspects* and they will not be undeleted. Two date and time fields are available. Local time and UTC time. This dataset has timezone -8 and -7. Switching to UTC is useful when multiple timezones are combined. Changing between these modes must be done with care in order to make sure the data presented is consistent with what the investigator is expecting.

The recovery function is able to recover a lot of messages. The ratio of recovered messages compared to the original count varies greatly among the accounts. Visualizing only the recovered messages in a time line would be an interesting approach. The manually recovered accounts found quite few messages, especially sent messages. It highlights the limitation of the method: messages must be still present in another users account. The mismatch between recovered messages and the total count found by using *grep* can be explained as duplicates where a message

sent to many people including our suspect is found in multiple accounts.

Many of the investigative questions have not been considered. Some of them require search capability like how to find all communication during a time window or based on content. Classification of message content in topic categories is a very interesting idea that has not been the focus of this thesis. The same goes for classification of user roles such as information hubs, interconnectivity of users and subgroups. The background chapter has a few examples of methods trying to answer these questions.

4.2 Experiment 2: Outlook PST file

4.2.1 Test setup

A valid PST file was required in order to test the conversion from *PST* files to a directory of *EML* files. Microsoft Outlook 2010 was installed and the authors Gmail account added using the IMAP protocol. IMAP is a synchronization protocol where the server is in control of the email account, similar to how mail clients set up with exchange function. The POP protocol by contrast is used when the client is in control of messages. Downloaded messages are by default removed from the server.

Outlook downloads message headers first in order to give a quick preview of who the message is from, dates and the subject. It can be configured to download a local copy of each message including attachments as seen in Figure 53.

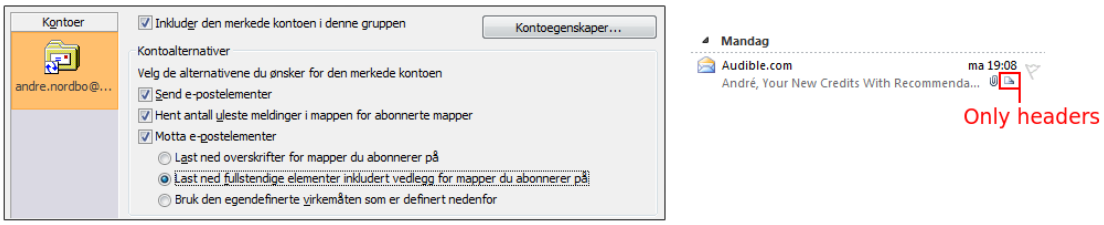


Figure 53: "*Download the entire message including attachments for subscribed folders*" loosely translated at left. The symbol indicating that only headers are downloaded for a message. Wait until all messages are synchronized before extracting the PST file.

Each account is stored in individual *PST* files and the path can be found by looking up the *properties* followed by *advanced* of the account. An example of path is shown in Figure 54.

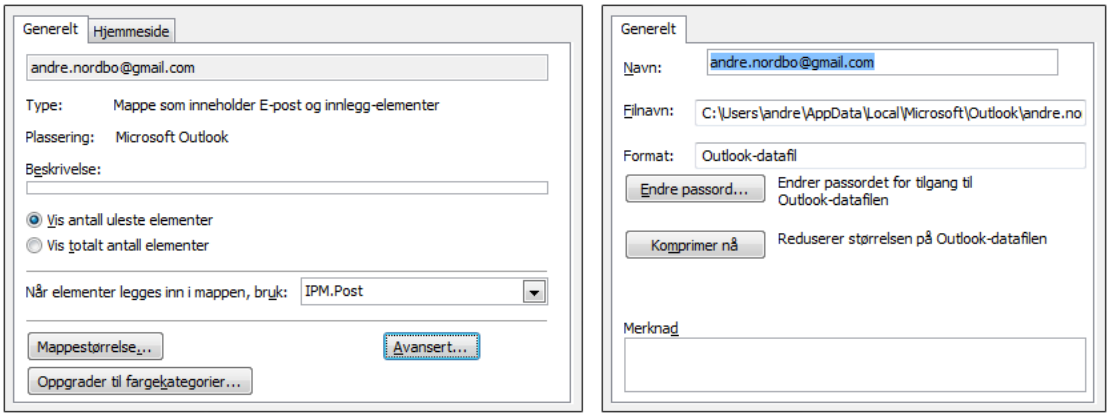


Figure 54: The properties of the Google Gmail account at left. Opening *advanced* at right where the full path is listed.

A copy of the PST file was transferred to the Ubuntu server. It was put in a folder *pst* inside a new project folder which was linked to in *settings.py*. The company domain was set to "@gmail.com" and file matching to *integer followed by .eml*. All steps in *runall.py* were enabled. Both *libpst* and the *Python OleFileIO_PL* libraries must be preinstalled. *libpst readpst* is used for extracting all *MSG* files and put them in a hierarchal folder structure. The root folder for every PST file is given the name of the account *email address* found inside the PST file. MSG is an internal storage format used by Microsoft and it is claimed impossible to get back exactly what was sent [88]. The original headers are available (as seen in Figure 55) so it is possible to reconstruct something very close as described in the Implementation chapter.

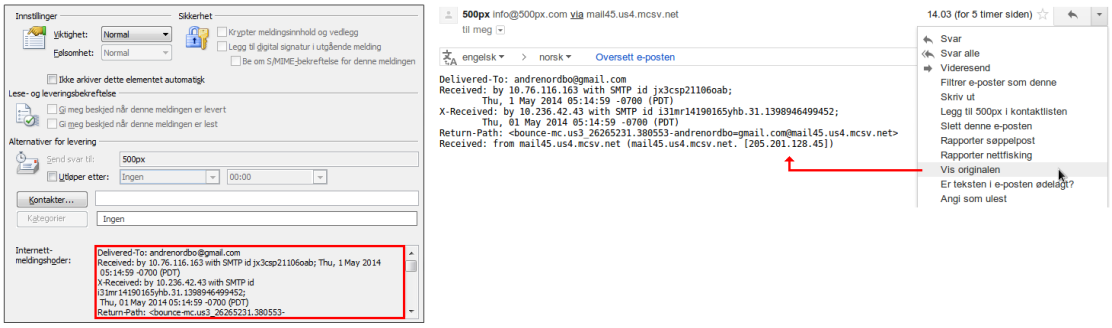


Figure 55: Left: The *Internet headers* for a selected message can be found by clicking *file*, *information* and *properties*. Right: Comparison of headers from Google Gmail using the *show original* function.

4.2.2 Results

Table 6, 7 and 5 summarize the main performance measurements:

Size of PST	Maildir size	Meta data size	Attachments	JSON content
1671 [MiB]	4096 [MiB]	1.7 [MiB]	2867 [MiB]	46.4 [MiB]

Table 5: Comparison of message and meta data size for PST file

Original messages	Duplicates removed	Messages left	Added back	Total messages
9,267	4,707	4,560	NA	4,560
100%	50.7%	49.2%	NA	

Table 6: Duplicate removal and undelete for PST file

System	Convert PST	Extract	Direction	Add back	Time statistics
Desktop	252 [sec]	235 [sec]	0.1 [sec]	0.5 [sec]	0.1 [sec]

Table 7: Speed of code for PST file. Note that both JSON files and attachments are written to disk during extraction of meta data.

The *dataset overview* is shown in Figure 56. Messages has been sent and received regularly, although the number of messages has increased. The account is dominated by a lot of direct messages while few forwards of information. The external simply means externally of a google.com address, and is less interesting in this case. 11 messages sent to suspects is simple messages sent by the owner to the owner.

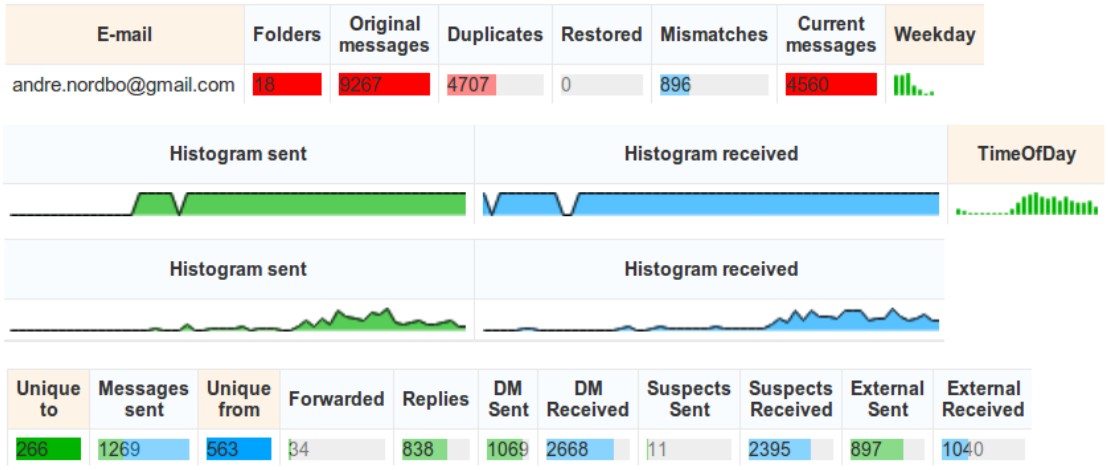


Figure 56: Gmail: Dataset overview

The *weekly timeline* and *time of day* plots indicate that sending messages is quite normal during weekends and that sending email during the night is not uncommon, although most activity seems to start between 10 and 12 in the morning. See Figure 57.

Figure 57: Gmail: Day of week and time of day

Figure 58 shows an example of HTML preview and attachments. An example of email with string encoding mismatch was also found, although very uncommon. It indicates more work is needed in PST file conversion. The attachment folder was also checked. Images are automatically previewed in Ubuntu even when the file extension is missing. Many duplicate images were found in the sense that only parts of the second copy would render.

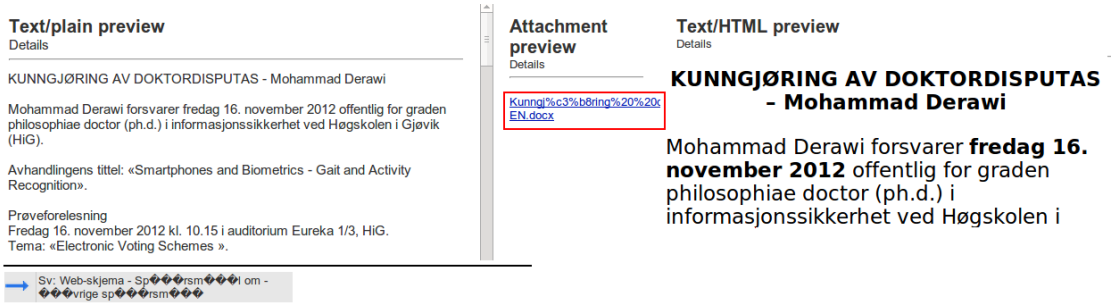


Figure 58: Gmail account: HTML preview and attachments

The user communication also contained some interesting patterns as seen in Figure 59:

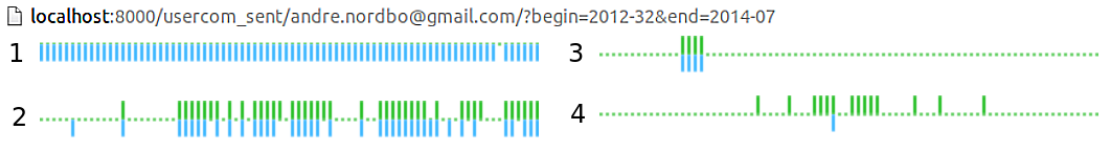


Figure 59: 1: Regular received messages resulting from subscription to a mailing list. 2: Regular two way communication. 3: A short burst of communication for a small period. 4: Many sent messages, few replies. The content does not require a reply and could be informative updates.

Looking at the *message detail view* with the draft flag sent revealed many messages sent to the author. Usage of the *delivered-to* instead of *To* field was the reason. Setting the mismatch flag revealed many messages from another email account. These messages result from moving messages across accounts using an email client called Thunderbird.

4.2.3 Discussions

The main goal of this experiment was to verify import of a PST file with recent email examples. Problems found with the current implementation are related to extraction of attachments and some examples of wrong decoding of character set. One account of almost 10,000 messages took approximately 8 minutes to prepare. It is this time period could be speed up considerable by modifying the readpst code to parse headers in a more forensically sound way. Storing each message as a plain text file consumes a lot of space on disk and is not very efficient. The JSON meta data file was almost 3% of the original PST file. The belonging JSON files with header and content is almost the same size as the PST file, and that is without the attachments, emphasizing once again the need for a space efficient way of storage.

Email messages can contain HTML content and previewing it can represent a major security issue. The HTML can contain code able to execute an attack of a known vulnerability. If given access to the Internet code can also download external resources notifying a 3rd party of access to the message. HTML versions are often accompanied by a plain text version for compatibility reasons, and could be used as an alternative solution to previewing the HTML content. A compromise must be made regarding the benefits of seeing the message as intended by the sender versus the risks involved. Separating HTML content in a dedicated iFrame avoids contamination by style sheet overrides and that JavaScript is executed separated from the code in the main interface.

4.3 Experiment 3: Tableau software comparison

4.3.1 Test setup

Tableau [89] is a commercial general purpose tool for connecting to, visualizing and sharing data [90]. Data is separated into *dimensions* and *measures* automatically. Dimensions are descriptive fields while measures are numerical data [91]. Dimensions and measures can be dragged into columns and rows, used for coloring and point size. Time data is treated specially allowing for custom aggregations such as yearly, monthly and quarterly. Similarly applies for locational data such as country, city or zip code. The software is available as desktop, server and online versions. Both server and online versions allow sharing of web based dashboards with multiple interactive visualizations so that other users can view and ask questions to the data. The online version is a hosted alternative to a self maintained server. Tableau Software has released a "visual analysis best practices" guidebook [92] emphasizing the importance of question-based visualizations. The whitepaper has many examples of how to select the appropriate chart type, prioritize and limit overload of data.

A free one year license of Tableau Desktop is available for students. Tableau does not support native import of email in any way but it does support import from Microsoft Excel and text files using comma separated values. Saving meta data from the parser step in this format was trivial using the default CSV library in Python. Each individual meta data JSON file was saved in a combined CSV-file. Tableau does not understand timestamps so both local and UTC timestamps were converted to string representation of the form: YYYY-mm-dd HH:MM:SS. An additional field *csv_owner* was added. The reason is that the existing *meta_username* reflects where the message was found, not the accounts actual username. This is a consequence of the recovery step. This program is named *tableau.py*.

Tableau failed to load the CSV file since comma separators collided with the decimal symbol, either in the software or as defined in the operating system. A *vertical bar* (also known as "pipe") was used instead for separation of fields. Inspection of the data revealed many integer values converted to decimal values. Dates and text was categorized as dimensions and the remaining numeric values as measures. A brief explanation of the interface is given in Figure 60.

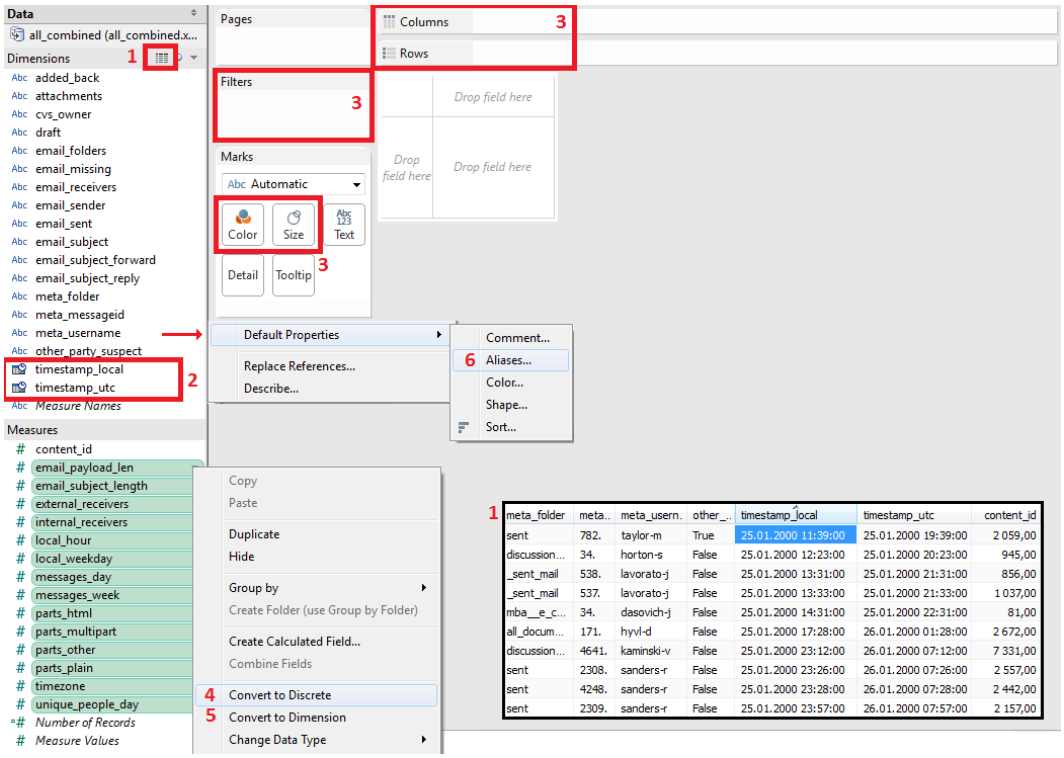


Figure 60: 1: Inspection of the imported data. 2: Date and time has been properly recognized. Make note of the the separation into dimensions and measures. 3: Data can be dragged on to *columns*, *rows*, *color* and *size*. Data can also be dragged to the *filters area*. 4 and 5: Change between discrete and continuous interpretation of the data. Make *measures* into *dimensions*. 6: Give alternative names to values such as true and false.

4.3.2 Results

The goal was to recreate visualizations from the web implementation using Tableau, and the first visualization recreated was a timeline of activity for all user accounts: *timestamp_local* was dragged onto *columns* and set to *daily mode*. The *cvs_owner* was dragged onto *rows*. The automatic mark type decided that *gantt bar* was fitting.

Timezone was also dragged onto *color*. A blue color was assigned to timezone of -8 and orange for -7. Aggregated data can contain both of these values in the transition period, colored in green. No other timezones were present in the whole dataset. This visualization is shown in Figure 61.

Figure 61: Period of activity for individual users in a binary mode. Time zone patterns are consistent across all accounts.

Replacing timezone with the *added_back* (recovered) flag and also adding it to the *Rows* yields another interesting look at the data as seen in Figure 62. Timelines are shown for every account split by original message and recovered message. A gap in original messages filled with recovered messages could be worth investigating.

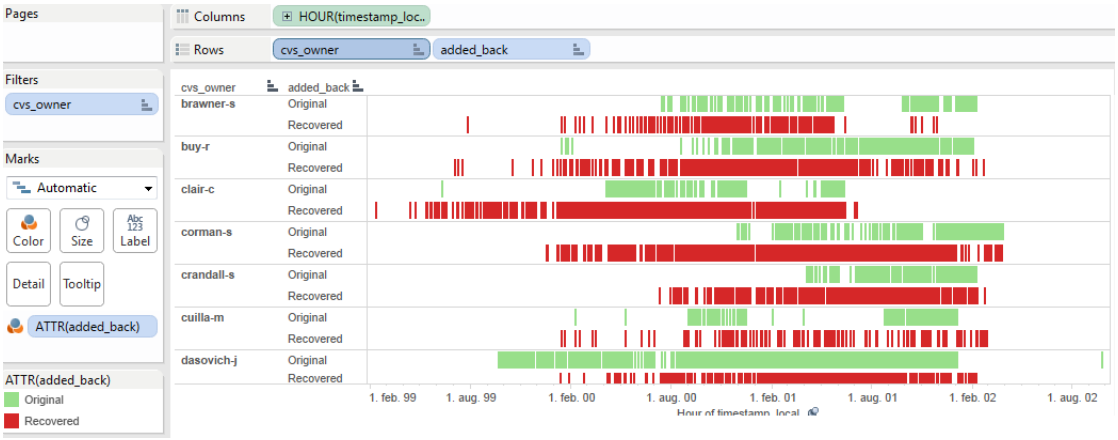


Figure 62: Recovered messages in red and original messages in green.

The next visualization recreated was a weekly aggregate of messages stacked by weekday and separated into the sending and received direction. The quick filter of *csv_owner* is used for selecting a single account. The *timestamp_local* was set in weekly aggregation mode for the columns. The attributes *csv_owner*, *email_sent* and a count of *content_id* were added to the rows. The weekly time aggregate was also added to color in discreet mode. *aliasing* was used for naming the directions properly. Mark type was set to *bar*. The resulting visualization is shown in Figure 63.

Figure 63: Tableau compared to the corresponding graphs in the web visualization implementation. The selected account is *taylor-m* for both.

A time of day plot can be drawn by dragging date as unique days on the *Columns*. On the *Rows* we put *email_sent*, *cvs_owner* and a combination field consisting of the hour of day + minute of day divided by 60. It was colored by the *added_back* attribute. See Figure 64.

Figure 64: Time of day for the user guzman-m. Splittet in sent and received direction and colored by original or recovered. Notice the area of recovered sent messages. Regular patterns are easily visible on the chart. The formula used to merge hour with minute is shown at the top.

4.3.3 Discussions

A major benefit of Tableau is how easy it is to experiment with various combinations of visualizations and filtering methods. A few mouse clicks is enough if you know what you want to achieve. A question raised with the *time of day* plots in experiment 1 was whether there are more timezones present than the observers -8 and -7. This question was very easy to answer by dragging the timezone onto *color*. The time of day plot was also easily extended to show the distribution of recovered recovered messages both in the sending and receiving direction.

A downside is that Tableau does not currently have any support for visualizing graph data. Visualizations can be created quickly, but many combinations of data make little or no sense. One option is to use Tableau server for sharing reconfigured *dashboards*. Another approach is to experiment with data using a solution such as Tableau and then implement configurations found useful into a web framework like the one proposed in this thesis. The benefit of that approach is to keep the power of interlinked accounts and easy access to the underlying data.

4.4 Anomaly detection summary

A naive approach to automatic anomaly detection is to convert email meta data into to numeric values, run a distance based anomaly detection method on it and see what happens. Many of the attributes extracted are numerical and can be used directly. Choosing attributes at random, or choosing whatever is available not a very good idea. Interpretation of the classified anomalies would be very hard and adding attributes of low relevance could degrade the resulting anomaly score, and a classified anomaly does not imply it is relevant for the investigation. The important question is: *abnormal compared to what?*[47]. What is normal.

A message can have a long subject or no subject. It can have attachments or no attachments. It can be sent to 1 person or many. These measures by themselves are not good indicators for an abnormal email message. The same argument holds for replies and forwarding of messages.

The approach taken in this thesis was to look for indicators of anomalous behavior using visual techniques. The important measurements are those closely related to the individual behind the account. Features based on time and participants in addition to deduced knowledge of content and the author. Measures that might work include:

- Sent messages at irregular weekdays. Irregular could be weekends for a corporate account, while normal for a personal account.
- Sent messages at unusual hours. Irregular could be outside of normal working hours, during the early morning hours or during night[2] depending on what is normal for that particular user or the team the user works in.
- Messages dated outside of expected date ranges. In the future or too far back in time.
- Messages recovered based on other users accounts could be suspicious in cases where it is suspected the owner has tried to get rid of evidence.
- A message directed to other suspects assuming a subset of possible accounts have been acquired.
- A measure of how likely it is a message is actually written by the author. A low value would be suspicious
- If messages could be classified based on topic then a less likely topic would be interesting.
- Messages sent to infrequent email addresses.
- An unusual high or low number of messages over a short period.

[2]Testing on unusual times of day would be difficult with the Enron dataset since hour of day is not consistent across the collected period.

5 Conclusions and further work

The main problem statement motivating this work was that current tools for supporting analysis of data in digital forensic investigations scale poorly with the increasing amount of data, spread across multiple sources. This thesis has researched an alternative approach to the contemporary disk-image based solutions typically followed in commercial forensic software suites. The alternative approach is based on the extraction of key properties from a chosen data source, email in this case, and using web technology for interlinking and visualizing various configurations of the properties. Properties such as time and involved participants. Moving the data and application from desktop applications to a web service also enables new ways of digital collaboration.

5.1 Research questions answered

7 research questions were defined in the introduction and used for guiding the summary of findings.

- Q1: What visualization techniques exists for aiding digital investigations in published literature?
- Q2: What (meta) data and preprocessing are being used to drive these visualization techniques?

The first two questions were targeted at getting a broad overview in the field of research. Techniques directly related to email include visualization of connectivity graphs in various forms and time lines. The important thing is how meta data is extracted and preprocessed. Visual connectivity graphs can have edges and nodes colored and sized according to centrality measures or message frequency. Messages can be clustered or classified, both in terms of topic and based on authorship. This information can further be used for filtering and visualizing new subsets of the data and reveal new insight. Timelines can be based on individual time observations, but more interesting is plotting aggregates and time versus other properties of the data. Especially when comparing multiple sources at the same time.

- Q3: What questions would a forensic expert want to answer from collected email?

Questions can be based on the participants involved, date and time and message content. The content can be broken down to the written content and the writing style. Examples of questions can be related to finding messages in a given time frame, messages at particular times of day, persons well or poorly connected and messages discussing a particular topic. An initial list of questions is enumerated on page 44.

The author would recommend surveying experts from relevant organizations such as Økokrim and Kripos in order to expand the understanding of what questions are relevant. It is important to get an understanding of what evidence look like. Is it totally dependent on the case or can meta data be a back channel for finding it?

- Q4: What functionality is needed of a framework for supporting research in email?

Data visualizations are dependent on the quality of the data, emphasizing the importance of thorough preprocessing. A framework should support any email format. It implies the framework must be modular, based on the verity of storage formats in existence. The proposed solution is to standardize on messages in plain text using the RFC 2822/MIME format, and extract individual meta data files per user account based on this baseline. Important preprocessing stages are duplicate removal, determination of account email address and determination of sending direction. Messages that has been deleted can be recovered if they still remain as sent or received in other email accounts. This technique can also be applied for partly recovery of missing email accounts. Important finding related to meta data extraction are:

- Extracted meta data must contain a reference to the original source.
- Determination of direction is important because sent messages are direct actions of the user. Knowing the fact that a message has been recovered from another account can be used for interesting timeline visualizations as shown in Figure 64 (page 84).
- Marking messages that does not contain the email address of the user, either sent or received, can help locating messages from alternative email accounts.
- Marking messages sent to or received from another suspect can be valuable in terms of filtering narrowing down the number of relevant messages. This is especially valuable in situations where only a few accounts have been acquired in large organizations.
- The number of recipients of a message can also be used for filtering. Messages sent to many people can be assumed less important in some investigations based on the rationale that a person is less likely to send sensitive information to a large group of people in fear of exposure.

Web technology can be used for both interaction and visualization. Important functionality is basing visualizations and message details on the meta data, while loading the actual content data only when needed. Hyperlinking is an efficient way of linking accounts and time periods. Messages read by an investigator can be tracked in a database so that other investigators can see what has been read. A simple case timeline is also proposed as a solution for adding messages of interest and keeping a shared summary of important findings.

This approach is similar to existing commercial e-discovery solutions, but differs in that data visualizations was the main focus, not indexing and searching. The commercial tools are closed source and very expensive. An open framework was thus needed for supporting this work and feature research in reaching the main goal. Empowering forensic investigators to comprehend and correlate digital evidence in an increasingly interconnected digital world.

- Q5: Can the investigative questions aid design of interesting visualizations?

A selection of questions were used for experimenting with various ways of visualizing email meta data. These visualizations include:

- *Weekly timeline*: Message count for every day of the week stacked weekly, separated in sending and receiving direction. Days are color coded with distinct colors for Saturday and Sunday, emphasizing weekends. The same visualization can be based on other counts such as unique *sent to* and *received from* addresses.
- *Time of day*: Plotting the hour of day versus time can give interesting insights into work patterns. It is important to separate between sent and received messages. Color coding using dimensions such as being recovered, forwarded, replied and message size can further narrow down on interesting patterns and anomaly behavior.
- *Communication breakdown*: Every individual stream of messages to and from another account can be visualized using histograms in a *small multiples* configuration. Separating in sent and received direction as seen in Figure 59 (page 79) enables quick determination of how regular and symmetric communication is.

- Q6: In what ways can an implementation be benchmarked?

The principle ways determined from the literature review can be summarized as: Efficiency benchmarking in terms of speed and/or precision in solving case scenarios, asking experts' opinions using questionnaires and lastly, qualitative case studies showing how an implemented technique or tool can be useful.

The last technique was chosen because of time constrains and the realization that defining meaningful case scenarios tend to be biased towards what the researcher wants to highlight. The chosen approach consisted of using the Enron dataset as a case study, both during the experimental development phase and for showing what the implemented visualizations look like with real data. The second experiment was using an up to date email account in Outlook PST format for verification of meta data extraction. The last experiment demonstrated how a general purpose visualization tool called Tableau (commercial) can speed up the process of visualization configuration and design.

The main conclusion based on these experiments is that using web technology is very powerful for interlinking across user accounts, aggregated summaries, visualizations and down to the original message. Implementing and experimenting with the data was a tedious task. Importing the merged extracted meta data into a tool such as Tableau allowed for simple drag and drop of meta data into various visual configurations and can speed up this process considerably. The downside is that many configurations of data are not meaningful and the lack of interlinking. A recommended approach is to use tools such as Tableau for experimenting and then implement meaningful visualization configurations back into a web based framework.

- Q7: What are the benefits and limitations of anomaly detection methods in terms of aiding email forensics?

The rationale for assuming benefits of anomaly detection was based on the assumption that illegal behavior can be detected based on outlier characteristics of meta data. What is considered an anomaly can depend on context. The context can be for the user, for a larger group, and it can also be the location of the person or a period of the year such as vacation time.

Data visualization methods allow humans to spot anomalies manually. Both anomalies based on profiles (Appendix C) and individual messages (Figure 49, page 70) has been explored in this thesis. The benefit is that interpretation is clear. Using machine learning algorithms for automating detection of outliers is a promising idea, but it must be grounded on an understanding of what mechanisms cause meta data from relevant messages to stand out. This in order to select the relevant features based on what type of anomaly is being looked for.

5.2 Theoretical implications

The algorithm for determination of email address of an account is based on the assumption that the most frequent address in *From* fields is most likely the owners address. This assumption has been proven wrong for multiple accounts in the Enron dataset. A proposed solution for improvement is to utilize the fact that many subscription services are machine generated and follow regular patterns. These patterns can be found and excluded. Accounts where the owner has multiple email addresses could be located by means of the *missmach* flag, triggered when the determined email is not found in a given message.

The algorithm for recovery of messages is based on direction which is based on determination of email address. This algorithm checks other user accounts based on meta data. If the message was sent then only the various *recipient* fields must be checked. It the message was received, then both *sender* and *recipient* fields must be checked. This is illustrated in Figure 36 on page 57. The implication is that recovery of messages is not only dependent on the available messages in other account, but also the correct determination of email addresses.

5.3 Practical considerations

Previewing of HTML content is a potential security threat in cases where embedded code in included in an email. If the preview has access to Internet, then another problem is access requests to linked resources which could notify a 3rd person of access to the message.

Messages has been found containing garbage characters, indicating decoding error of character set. This is a consequence of the implementation being a simple prototype, and another related implication is that PST parsing is unnecessary slow due to massive calculations and writes to disk in vain when using *libpst*.

Time inconsistencies has been found in the Enron dataset by visualization of *time of day* and it has been verified that account have multiple email addresses valid for the owner. Care must be taken if the dataset is used for studying author attribution.

The Libpst library has been found unreliable for forensically sound extraction of messages from PST files, based on missing header informations and header transformations.

5.4 Further work

The goal of research is to answer a few questions and create many new ones. The work in this thesis is merely a humble start. Further research and development can be divided into 3 groups: Experimentation on expert investigators, extending the framework and optimization of code in case it is found beneficial.

5.4.1 Testing on expert investigators

An important task is to determine what benefit such a framework could give professional forensic investigators. One approach is to put the prototype in use in order to gather feedback on what functionality is of value.

What is more important than demonstrating that a developed method is useful, is to study forensic investigators in order to determine what they actually struggle with. What are the tedious repetitive tasks and manual correlation they have to perform? How can these tasks be speed up or automated?

5.4.2 Extending the framework

A preconfigured environment is built using VirualBox where all implementation code is made available. The primary reason for this choice is to cut down on the time needed to set up the system including the required dependencies. This virtual machine is made available at the forensic lab at Gjøvik University College. It consists of an installation of Ubuntu Desktop 14.04 with the username: *email* and password: *email*. Code and the Enron dataset is available inside together with short documentation on how to use it and required dependencies.

Research directions include:

- Multiple email addresses: How can the framework can be extended to deal with accounts having multiple email addresses?
- Content classification: Email messages has unstructured information in the form of text and file attachments. How can the unstructured content be analyzed for classifying semantic meaning or topic? Can sent messages in an account be ranked in terms of how likely they are actually sent from the account owner?
- Indexing and searching: How can indexing and searching be added to the framework? Splunk[1] is an interesting candidate, and other open source alternatives are Elastic Search[2] and Solar[3].
- Automatic anomaly detection: Can individual messages and accounts be scored in terms of how abnormal they are? One option is to build profiles for individuals and compare message attributes to the profile. The profile could be based on probability distribution for various measures such as time of day and day of week.

[1] Splunk at `http://www.splunk.com`
[2] Elastic Search at `http://www.elasticsearch.org`
[3] Solar at `http://lucene.apache.org/solr/`

- Additional meta data: What other meta data fields can be useful in investigations? Example could be using IP-addresses for plotting locations to a geographical map.
- Thread merging: How can message threads be merged, and is is useful?

5.4.3 Optimizing code

- Add parsers for other email client in order to support formats of other email clients.
- Safe preview of HTML: Make a secure HTML preview is not prone to malicious attacks and information leakage.
- Add or improve visualizations based on the meta data. Implementation of filters for visualizations such as recovered, forwarded and number of recipients. Add graph based visualizations.
- Graphical design and interaction: In what way can the web user interface be optimized? Examples can be new ways of interlinking information, size and location of buttons for control, keeping filters between pages and soring of columns.

Bibliography

[1] Franke, K. & Srihari, S. 2008. Computational forensics: An overview. In *Computational Forensics*, Srihari, S. & Franke, K., eds, volume 5158 of *Lecture Notes in Computer Science*, 1–10. Springer Berlin Heidelberg.

[2] Osborne, G. & Turnbull, B. 2009. Enhancing computer forensics investigation through visualisation and data exploitation. In *Availability, Reliability and Security, 2009. ARES '09. International Conference on*, 1012–1017.

[3] AskDefine. Forensic. http://forensic.askdefine.com/. (Visited Jan. 2014).

[4] Raghavan, S. 2013. Digital forensic research: current state of the art. *CSI Transactions on ICT*, 1(1), 91–114.

[5] Bernstein, D. & Jackson, J. D. 2004. The daubert trilogy in the states. *Jurimetrics*, 44. Available at SSRN: `http://ssrn.com/abstract=498786`.

[6] Kaufman, H. 2001. The expert witness. neither frye nor daubert solved the problem: what can be done? *Science and Justice*, 41(1), 7 – 20.

[7] Garfinkel, S. L. 2010. Digital forensics research: The next 10 years. *Digital Investigation*, 7, Supplement(0), S64 – S73. The Proceedings of the Tenth Annual {DFRWS} Conference.

[8] Schrenk, G. & Poisel, R. 2011. A discussion of visualization techniques for the analysis of digital evidence. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, 758–763.

[9] Thesis presentation meeting at Økokrim. 14. March 2014, The supervisor of this thesis arranged a meeting with Økokrim the 14th March 2014 with presentation of current progress and discussions regarding tools and data formats. (Not published).

[10] Wilberg, L. 2014. Presentation in imt4012 digital forensics 1: Digital investigations in the private sector.

[11] Garfinkel, S. 2012. Lessons learned writing digital forensics tools and managing a 30tb digital evidence corpus. *Digital Investigation*, 9, Supplement(0), S80 – S89. The Proceedings of the Twelfth Annual DFRWS Conference.

[12] Richard, G. & Roussev, V. 2006. Digital crime and forensic science in cyberspace. chapter Digital Forensics Tools: The Next Generation, pp.75–90. Idea Group Publishing.

[13] Ariu, D., Giacinto, G., & Roli, F. 2011. Machine learning in computer forensics (and the lessons learned from machine learning in computer security). In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, 99–104, New York, NY, USA. ACM.

[14] Radicati, S. & Hoang, Q. 2011. Email statistics report, 2011-2015. http://www.radicati.com/wp/wp-content/uploads/2011/05/Email-Statistics-Report-2011-2015-Executive-Summary.pdf. The Radicati Group, inc.

[15] 2013. International center for integrative systems: History of email. http://www.inventorofemail.com/history_of_email.asp#top_of_the_page. Last visited: May 2014.

[16] J. Haggerty, D. L. & Taylor, M. Social network visualization for forensic investigation of e-mail. Proceedings of the Fourth International Workshop on Digital Forensics and Incident Analysis (WDFIA 2009).

[17] Kent, K., Chevalier, S., Grance, T., & Dang, H. 2006. Guide to integrating forensic techniques into incident response. http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf. NIST Special Publication 800-86 (Page 3-1).

[18] Endicott-Popovsky, B., Frincke, D. A., & Taylor, C. A. 2007. A theoretical framework for organizational network forensic readiness. *JOURNAL OF COMPUTERS*, 2(3).

[19] Baker, S., Falkenrath, R., Cole, D., & German, M. 2013. Spy on me, i'd rather be safe. http://intelligencesquaredus.org/debates/past-debates/item/987-spy-on-me-id-rather-be-safe. Intelligence Squared U.S. debate.

[20] Carrier, B. D. & Spafford, E. H. 2004. An event-based digital forensic investigation framework. In *In Proceedings of the 2004 Digital Forensic Research Workshop*.

[21] Goodman, M. D. 1997. Why the police don't care about computer crime. http://jolt.law.harvard.edu/articles/pdf/v10/10HarvJLTech465.pdf. Harvard Journal of Law & Technology. Volume 10, Number 3. Page 469.

[22] Carter, D. L. 1995. Computer crime categories. FBI Law Enforcement Bulletin; Jul95, Vol. 64 Issue 7, p21.

[23] Alkaabi, A., Mohay, G., McCullagh, A., & Chantler, N. 2011. Dealing with the problem of cybercrime. In *Digital Forensics and Cyber Crime*, Baggili, I., ed, volume 53 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 1–18. Springer Berlin Heidelberg.

[24] Sherman, C. The senses — a primer (part ii). http://www.brainfacts.org/sensing-thinking-behaving/senses-and-perception/articles/2013/the-senses-a-primer-part-ii/. Creation Date: 25 Sept 2013 (Last visited May 2014).

[25] Roussas, G. Visualization of client-side web browsing and email activity. Master's thesis, Naval Postgraduate School, Monterey, California, 2009.

[26] Dictionary.com. Visualize. `http://dictionary.reference.com/browse/visualize`. Last visited May 2014.

[27] McCormick, B., DeFanti, T., & Brown, M. 1987. Visualization in scientific computing. *Computer Graphics*, 21(6).

[28] Aigner, W., Miksch, S., Schumann, H., & Tominski, C. 2011. *Visualization of Time-Oriented Data*. Springer.

[29] Aigner, W., Bertone, A., & Miksch, S. 2007. Tutorial: Introduction to visual analytics. In *HCI and Usability for Medicine and Health Care*, Holzinger, A., ed, volume 4799 of *Lecture Notes in Computer Science*, 453–456. Springer Berlin Heidelberg.

[30] Friendly, M. 1995. Milestones in the history of thematic cartography, statistical graphics, and data visualization. In *13th international conference on database and expert systems applications (DEXA 2002), aix en provence*, 59–66. Press.

[31] Garfinkel, S. November 2013. Forensics visualizations with open source tools (presentation). `http://www.basistechweek.com/osdf.html`. 4th annual Open Source Digital Forensics Conference.

[32] Shneiderman, B. 1996. The eyes have it: A task by data type taxonomy for information visualizations. In *In IEEE Symposium on Visual Languages*, 336–343.

[33] Yaffa, J. The information sage. `http://www.washingtonmonthly.com/magazine/mayjune_2011/features/the_information_sage029137.php?page=1#`. Published May/June 2011 (Last visited: May 2014).

[34] Tufte, E. R. 1983. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut.

[35] Tufte, E. R. 1990. *Envisioning Information*. Graphics Press, Cheshire, Connecticut.

[36] Tufte, E. R. 1997. *Visual Explanations Images and Quantities, Evidence and Narrative*. Graphics Press, Cheshire, Connecticut.

[37] Heer, J., Bostock, M., & Ogievetsky, V. June 2010. A tour through the visualization zoo. *Commun. ACM*, 53(6), 59–67.

[38] Heer, J. A brief history of data visualization. `http://hci.stanford.edu/courses/cs547/abstracts/08-09/090213-heer.html`. Last visited: May 2014.

[39] Keim, D., Kohlhammer, J., Ellis, G., & Mansmann, F., eds. *Mastering the Information Age: Solving Problems with Visual Analytics*. VisMaster, http://www.vismaster.eu/book/, 2010.

[40] 2013. Digital forensics ii: Lab sessions with jeff hamm. http://english.hig.no/content/view/full/25727/language/eng-US. Not published.

[41] Enderlein, G. 1987. Hawkins, d. m.: Identification of outliers. chapman and hall, london – new york 1980, 188 s., £ 14, 50. *Biometrical Journal*, 29(2), 198–198.

[42] Kononenko, I. & Kukar, M. 2007. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited.

[43] Forster, M. R. Notice: No free lunches for anyone, bayesians included. http://philosophy.wisc.edu/forster/papers/Krakow.pdf. Philosophy Department, University of Wisconsin, Madison (Last visited May 2014).

[44] Watanabe, S. 1969. *Knowing and guessing: a quantitative study of inference and information*. Wiley.

[45] Nguyen, H., Franke, K., & Petrović, S. 2012. Reliability in a feature-selection process for intrusion detection. In *Reliable Knowledge Discovery*, Dai, H., Liu, J. N. K., & Smirnov, E., eds, 203–218. Springer US.

[46] Chandola, V., Banerjee, A., & Kumar, V. July 2009. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), 15:1–15:58.

[47] Franke, K. Master thesis supervision meetings. Not published.

[48] Hargreaves, C. & Patterson, J. 2012. An automated timeline reconstruction approach for digital forensic investigations. *Digital Investigation*, 9, Supplement(0), S69 – S79. The Proceedings of the Twelfth Annual {DFRWS} Conference 12th Annual Digital Forensics Research Conference.

[49] Garfinkel, S., Nelson, A. J., & Young, J. 2012. A general strategy for differential forensic analysis. *Digital Investigation*, 9, Supplement(0), S50 – S59. The Proceedings of the Twelfth Annual {DFRWS} Conference 12th Annual Digital Forensics Research Conference.

[50] Simson L. Garfinkel, Alex Nelson, J. Y. August 2012. A general strategy for differential forensic analysis (presentation). Naval Postgraduate School.

[51] Olsson, J. & Boldt, M. 2009. Computer forensic timeline visualization tool. *Digital Investigation*, 6, Supplement(0), S78 – S87. The Proceedings of the Ninth Annual {DFRWS} Conference.

[52] Mackinlay, J. D., Robertson, G. G., & Card, S. K. 1991. The perspective wall: detail and context smoothly integrated. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, 173–176, New York, NY, USA. ACM.

[53] Apperley, M. Bifocal display. http://www.interaction-design.org/encyclopedia/bifocal_display.html.

[54] Waloszek, G. Perspective wall. http://www.sapdesignguild.org/editions/highlight_articles_04/controls/PerspectiveWall.htm.

[55] Leschke, T. R. & Sherman, A. T. 2012. Change-link: A digital forensic tool for visualizing changes to directory trees. In *Proceedings of the Ninth International Symposium on Visualization for Cyber Security*, VizSec '12, 48–55, New York, NY, USA. ACM.

[56] Flaglien, A. O. Cross-computer malware detection in digital forensics. Master's thesis, Gjøvik University College, Gjøvik, Norway, 2010.

[57] Flaglien, A., Franke, K., & Arnes, A. 2011. Identifying malware using cross-evidence correlation. In *Advances in Digital Forensics VII*, Peterson, G. & Shenoi, S., eds, volume 361 of *IFIP Advances in Information and Communication Technology*, 169–182. Springer Berlin Heidelberg.

[58] Fei, B. K. L. Data visualisation in digital forensics. Master's thesis, 2007. Supervised by Prof. J.H.P. Eloff at University of Pretoria, South Africa.

[59] Lowman, S. & Ferguson, I. Web history visualisation for forensic investigations. Technical report, 2010.

[60] Wright, C. V., Monrose, F., & Masson, G. M. 2006. Using visual motifs to classify encrypted traffic. In *Proceedings of the 3rd international workshop on Visualization for computer security*, VizSEC '06, 41–50, New York, NY, USA. ACM.

[61] Garfinkel, S. L. November 2013. Forensics visualizations with open source tools. http://www.basistechweek.com/osdf.html#speaker-simson-garfinkel. 4th Annual Open Source Digital Forensics conference.

[62] Axelsson, S. *Understanding Intrusion Detection Through Visualization*. PhD thesis, Göteborg University, Göteborg, Sweeden, 2005.

[63] Laclavík, M., Seleng, M., Kvassay, M., Gatial, E., & Balogh, Z. 2010. Email analysis and information extraction for enterprise benefit. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.186.7126.

[64] de Vel, O., Anderson, A., Corney, M., & Mohay, G. 2001. Mining e-mail content for author identification forensics. *SIGMOD RECORD*, 30, 55–64.

[65] Heer, J. 2005. exploring enron: Visualizing ANLP results: – a course project. http://jheer.org/enron/v1/.

[66] Hansen, D., Shneiderman, B., & Smith, M. 2010. Visualizing threaded conversation networks: Mining message boards and email lists for actionable insights. In *Active Media Technology*, An, A., Lingras, P., Petty, S., & Huang, R., eds, volume 6335 of *Lecture Notes in Computer Science*, 47–62. Springer Berlin Heidelberg.

[67] Henseler, H. 2010. Network-based filtering for large email collections in e-discovery. *Artificial Intelligence and Law*, 18(4), 413–430.

[68] Haggerty, J., Karran, A. J., Lamb, D. J., & Taylor, M. J. 2011. A framework for the forensic investigation of unstructured email relationship data. *IJDCF*, 3(3), 1–18.

[69] Niessen, M. December 2012. Ibm i2 analyst's notebook - esri edition (presentation). `http://www.youtube.com/watch?v=MJ5CovDQDYU`. YouTube demonstration of software.

[70] Jankun-Kelly, T., Franck, J., Wilson, D., Carver, J., Dampier, D., & Swan, J.Edward, I. 2008. Show me how you see: Lessons from studying computer forensics experts for visualization. In *Visualization for Computer Security*, Goodall, J., Conti, G., & Ma, K.-L., eds, volume 5210 of *Lecture Notes in Computer Science*, 80–86. Springer Berlin Heidelberg.

[71] Coffee, J. C. July 2002. Understanding enron: It's about the gatekeepers, stupid. Columbia Law & Economics Working Paper No. 207.

[72] Grieve, T. The decline and fall of the enron empire. `http://www.salon.com/2003/10/14/enron_22/`. Published Tuesday, Oct 14, 2003 (Last visited May 2014).

[73] Custodian names and titles. `http://enrondata.org/content/custodians-names-and-title/`. Published January 17, 2009 (Last visited May 2014).

[74] Cohen, W. W. Enron email dataset. `http://www.cs.cmu.edu/~./enron/`. (Visited May. 2014).

[75] Klimt, B. & Yang, Y. 2004. The enron corpus: A new dataset for email classification research. In *Machine Learning: ECML 2004*, Boulicaut, J.-F., Esposito, F., Giannotti, F., & Pedreschi, D., eds, volume 3201 of *Lecture Notes in Computer Science*, 217–226. Springer Berlin Heidelberg.

[76] Resnick), I. E. P. August 1982. Rfc822: Standard for the format of ARPA internet text messages. https://tools.ietf.org/html/rfc822. Page 21 (Visited May. 2014).

[77] by David H. Crocker), I. R. April 2001. Rfc2822: Internet message format. https://tools.ietf.org/html/rfc2822. (Visited May. 2014).

[78] Weka: The university of waikato: Arff. http://weka.wikispaces.com/ARFF. (Visited May. 2014).

[79] Bodkin, R. Real-time big data analytics with storm. http://www.youtube.com/watch?v=hVO5nbxnBkU. CEO of Think Big Analytics (Visited May. 2014).

[80] Microsoft. Create an outlook data file (.pst) to save your information. http://office.microsoft.com/en-001/outlook-help/create-an-outlook-data-file-pst-to-save-your-information-HA010355677.aspx. (Visited May. 2014).

[81] Patricio, A. December 2013. Managing pst import-export process in exchange server 2013 (part 1). http://www.msexchange.org/articles-tutorials/exchange-server-2013/management-administration/managing-pst-import-export-process-exchange-server-2013-part1.html. (Visited May. 2014).

[82] Scarazzo, J. R. April 2011. Exchange message tracking logs - message forensics. http://www.ftitechnology.com/News-Events/Media-Coverage/BNA,-inc.-041811.aspx. (Visited May. 2014).

[83] Smith, D. libpst utilities. http://www.five-ten-sg.com/libpst/. (Visited May. 2014. Version 0.6.63).

[84] Lagadec, P. Olefileio_pl 0.30. https://pypi.python.org/pypi/OleFileIO_PL/. (Visited May. 2014. Version 0.30).

[85] Microsoft. [ms-oxmsg]: Outlook item (.msg) file format. http://msdn.microsoft.com/en-us/library/cc463912.aspx. (Visited May. 2014).

[86] Fiskerstrand, P. fileformat.info: Outlook msg file format. http://www.fileformat.info/format/outlookmsg/index.htm. (Visited May. 2014).

[87] EnronData. Enron custodians information. http://enrondata.org/assets/edo_enron-custodians-data.html. Last visited May. 2014.

[88] Ben. Stackoverflow: Convert pst into multiple eml files in c#. http://stackoverflow.com/questions/14154492/convert-pst-into-multiple-eml-files-in-c-sharp. (Visited May. 2014).

[89] Software, T. Tableau software: Business intelligence and analytics. http://www.tableausoftware.com/. Last visited May 2014.

[90] Software, T. Tableau software overview. http://www.youtube.com/watch?v=Z5kQR71yJpE. (Last visited May 2014).

[91] Software, T. Tableau version 8: Product demo. http://www.youtube.com/watch?v=bbEA1PsbOm4. original title: tableau-8-product-demo-final (Last visited May 2014).

[92] Software, T. Visual analysis best practices: Simple techniques for making every data visualization useful and beautiful. http://www.tableausoftware.com/learn/whitepapers/tableau-visual-guidebook. Last updated March 19, 2014.

[93] Walker, M. Github: msg-extractor. https://github.com/mattgwwalker/msg-extractor. (Visited May. 2014).

[94] Koryak, M. jquery.floatthead: A floating/locked/sticky table header plugin. http://mkoryak.github.io/floatThead/. Last visited May. 2014.

[95] Watts, G. jquery sparklines: jquery plugin for generating sparklines. http://omnipotent.net/jquery.sparkline. Last visited May. 2014.

[96] de Jong, J. Chap links library: Timeline. http://almende.github.io/chap-links-library/js/timeline/doc/. Last visited May. 2014.

[97] Bostock, M., Ogievetsky, V., & Heer, J. December 2011. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12), 2301–2309.

[98] Re-usable charts for d3.js. http://nvd3.org/. (Visited May. 2014).

A Missing headers using libpst

Original

```
Delivered-To: andre.nordbo@gmail.com
Received: by 10.205.64.12 with SMTP id xg12csp48850bkb;
        Mon, 3 Mar 2014 01:46:05 -0800 (PST)
X-Received: by 10.15.49.65 with SMTP id i41mr9881314eew.87.1393839964988;
        Mon, 03 Mar 2014 01:46:04 -0800 (PST)
Return-Path: <xxx@hig.no>
Received: from mail.stud.hig.no (mail.stud.hig.no. [2001:700:1d00:17::146])
        by mx.google.com with ESMTPS id x47si6969511eel.139.2014.03.03.01.46.04
        for <andre.nordbo@gmail.com>
        (version=TLSv1.2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);
        Mon, 03 Mar 2014 01:46:04 -0800 (PST)
Received-SPF: pass (google.com: domain of xxx@hig.no designates 2001:700:1d00:17::146 as permitted
sender) client-ip=2001:700:1d00:17::146;
Authentication-Results: mx.google.com;
       spf=pass (google.com: domain of xxx@hig.no designates 2001:700:1d00:17::146 as permitted
sender) smtp.mail=xxx@hig.no
Received: by mail.stud.hig.no (Postfix)
        id DF86E6A918; Mon,  3 Mar 2014 10:45:57 +0100 (CET)
Delivered-To: xxxxxx@mail.stud.hig.no
Received: from smtp.hig.no (smtp.hig.no [IPv6:2001:700:1d00:17::135])
        (using TLSv1 with cipher ADH-AES256-SHA (256/256 bits))
        (No client certificate requested)
        by mail.stud.hig.no (Postfix) with ESMTPS id ADE9F5FE65;
        Mon,  3 Mar 2014 10:45:55 +0100 (CET)
Received: by smtp.hig.no (Postfix)
        id 5F65DBF809C; Mon,  3 Mar 2014 10:45:55 +0100 (CET)
Delivered-To: imt4904v14@hig.no
Received-SPF: Pass (sender SPF authorized) identity=mailfrom; client-ip=128.39.41.47;
helo=mail.ansatt.hig.no; envelope-from=xxx@hig.no; receiver=imt4904v14@hig.no
Received: from mail.ansatt.hig.no (ansatt.hig.no [128.39.41.47])
        (using TLSv1 with cipher RC4-MD5 (128/128 bits))
        (No client certificate requested)
        by smtp.hig.no (Postfix) with ESMTP id 40499BF8081
        for <IMT4904V14@hig.no>; Mon,  3 Mar 2014 10:45:52 +0100 (CET)
Received: from mail.ansatt.hig.no ([128.39.41.47]) by mail.ansatt.hig.no
 ([128.39.41.47]) with mapi; Mon, 3 Mar 2014 10:45:53 +0100
From: xxx <xxx@hig.no>
To: "IMT4904V14@hig.no" <IMT4904V14@hig.no>
CC: xxx <xxx@hig.no>, xxx
        <xxx@hig.no>, "xxx" <xxx@hig.no>
Date: Mon, 3 Mar 2014 10:45:52 +0100
Subject: IMT4904 Master thesis 2014 / kandidatliste
Thread-Topic: IMT4904 Master thesis 2014 / kandidatliste
Thread-Index: Ac82w/FxXhaZ91ltRDqaDx5uzMDl4A==
Message-ID: <9C7C5854F550CA41945DF64623DEC587EECF0CED65@mail.ansatt.hig.no>
Accept-Language: en-US
Content-Language: nb-NO
X-MS-Has-Attach: yes
X-MS-TNEF-Correlator:
acceptlanguage: en-US
Content-Type: multipart/mixed;
        boundary="_004_9C7C5854F550CA41945DF64623DEC587EECF0CED65mailansatthig_"
MIME-Version: 1.0
--_004_9C7C5854F550CA41945DF64623DEC587EECF0CED65mailansatthig_
Content-Type: multipart/alternative;
        boundary="_000_9C7C5854F550CA41945DF64623DEC587EECF0CED65mailansatthig_"
--_000_9C7C5854F550CA41945DF64623DEC587EECF0CED65mailansatthig_
Content-Type: text/plain; charset="us-ascii"
Content-Transfer-Encoding: quoted-printable

Dear all...
```

Custom parsing

```
Content-Type: multipart/alternative;
 boundary="===============3477711606577949479=="
MIME-Version: 1.0
Delivered-To: andre.nordbo@gmail.com
Received: by 10.205.64.12 with SMTP id xg12csp48850bkb; Mon, 3 Mar 2014
 01:46:05 -0800 (PST)
X-Received: by 10.15.49.65 with SMTP id i41mr9881314eew.87.1393839964988; Mon,
 03 Mar 2014 01:46:04 -0800 (PST)
Return-Path: <xxx@hig.no>
Received-SPF: pass (google.com: domain of xxx@hig.no designates
 2001:700:1d00:17::146 as permitted sender) client-ip=2001:700:1d00:17::146;
Authentication-Results: mx.google.com; spf=pass (google.com: domain of
 xxx@hig.no designates 2001:700:1d00:17::146 as permitted sender)
 smtp.mail=xxx@hig.no
From: xxx <xxx@hig.no>
To: "IMT4904V14@hig.no" <IMT4904V14@hig.no>
CC: Slobodan Petrovic <slobodan.petrovic@hig.no>, Rune Hjelsvold
 <rune.hjelsvold@hig.no>, "Miriam E. Nes Begnum" <miriam.begnum@hig.no>
Date: Mon, 3 Mar 2014 10:45:52 +0100
Subject: IMT4904 Master thesis 2014 / kandidatliste
Thread-Topic: IMT4904 Master thesis 2014 / kandidatliste
Thread-Index: Ac82w/FxXhaZ91ltRDqaDx5uzMDl4A==
Message-ID: <9C7C5854F550CA41945DF64623DEC587EECF0CED65@mail.ansatt.hig.no>
Accept-Language: en-US
Content-Language: nb-NO
X-MS-Has-Attach: yes
X-MS-TNEF-Correlator:
acceptlanguage: en-US

--===============3477711606577949479==
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Content-Type: text/plain; charset="iso-8859-1"

Dear all...
```

LibPST readpst

```
Status: RO
From: "xxx" <xxx@hig.no>
Subject: IMT4904 Master thesis 2014 / kandidatliste
To: IMT4904V14@hig.no
Cc: xxx; xxx; xxx
Date: Mon, 03 Mar 2014 09:45:52 +0000
Message-Id:
<9C7C5854F550CA41945DF64623DEC587EECF0CED65@mail.ansatt.hig.no>
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="--boundary-LibPST-iamunique-1772372148_-_-"


----boundary-LibPST-iamunique-1772372148_-_-
Content-Type: multipart/alternative;
        boundary="alt---boundary-LibPST-iamunique-1772372148_-_-"

--alt---boundary-LibPST-iamunique-1772372148_-_-
Content-Type: text/plain; charset="Windows-1252"

Dear all...
```

B Automatic determination of email address

This listing is based on [87] and has been filled with the guessed email address and comments. It is sorted by folder name.

ID	FOLDER NAME	GUESSED EMAIL ADDRESS	FULL NAME	STATUS	TITLE	OTHER INFO
1	allen-p	phillip.allen@enron.com	Philip Allen	OK	N/A	
2	arnold-j	john.arnold@enron.com	John Arnold	OK	Vice President	
3	arora-h	harry.arora@enron.com	Harry Arora	OK	Vice President	
4	badeer-r	robert.badeer@enron.com	Robert Badeer	OK	Director	
5	bailey-s	susan.bailey@enron.com	Susan Bailey	OK	N/A	
6	bass-e	eric.bass@enron.com	Eric Bass	OK	Trader	
7	baughman-d	don.baughman@enron.com	Don Baughman	OK	Trader	
8	beck-s	sally.beck@enron.com	Sally Beck	OK	Employee	Chief Operating Officer
9	benson-r	robert.benson@enron.com	Robert Benson	OK	Director	
10	blair-l	lynn.blair@enron.com	Lynn Blair	OK	N/A	
11	brawner-s	sandra.brawner@enron.com	Sandra Brawner	OK	Director	
12	buy-r	rick.buy@enron.com	Rick Buy	OK	Manager	Chief Risk Management Officer
13	campbell-l	larry.campbell@enron.com	Larry Campbell	OK	N/A	
14	carson-m	mike.carson@enron.com	Mike Carson	OK	Employee	
15	cash-m	michelle.cash@enron.com	Michelle Cash	OK	N/A	
16	causholli-m	monika.causholli@enron.com	Monika Causholli	OK	Employee	Analyst Risk Management
124	clair-c	carol.clair@enron.com	Carol St. Clair	GUESS, OK	In House Lawyer	
17	corman-s	shelley.corman@enron.com	Shelley Corman	OK	Vice President	Regulatory Affairs
18	crandell-s	sean.crandall@enron.com	Sean Crandall	GUESS, OK	Director	
19	cuilla-m	martin.cuilla@enron.com	Martin Cuilla	OK	Manager	
20	dasovich-j	jeff.dasovich@enron.com	Jeff Dasovich	OK	Employee	Government Relation Executive
21	davis-d	dana.davis@enron.com	Dana Davis	OK	N/A	
22	dean-c	clint.dean@enron.com	Craig Dean	FAIL **craig.dean@enron.com** and clint.dean@enron.com	Trader	
23	delainey-d	david.delainey@enron.com	David Delainey	OK	CEO	Enron North America and Enron Enery Services
24	derrick-j	james.derrick@enron.com	James Derrick	OK	In House Lawyer	
25	dickson-s	stacy.dickson@enron.com	Stacy Dickson	OK	Employee	
26	donoho-l	lindy.donoho@enron.com	Lindy Donoho	OK	Employee	
27	donohoe-t	tom.donohoe@enron.com	Tom Donohoe	OK	N/A	
28	dorland-c	chris.dorland@enron.com	Chris Dorland	OK	Employee	
29	ermis-f	frank.ermis@enron.com	Frank Ermis	OK	Director	
30	farmer-d	daren.farmer@enron.com	Daren Farmer	OK	Manager	Logistics Manager
31	fischer-m	mary.fischer@enron.com	Mary Fischer	OK	Employee	
32	forney-j	m..forney@enron.com	John M. Forney	OK	Manager	Real time Trading Desk
33	fossum-d	drew.fossum@enron.com	Drew Fossum	OK	Vice President	
34	gang-l	lisa.gang@enron.com	Lisa Gang	OK	N/A	Senior Specialist
35	gay-r	randall.gay@enron.com	Randall Gay	OK	N/A	
36	geaccone-t	tracy.geaccone@enron.com	Tracy Geaccone	OK	Employee	
37	germany-c	chris.germany@enron.com	Chris Germany	OK	Employee	
38	gilbertsmith-d	doug.gilbert-smith@enron.com	Douglas Gilberth-Smith	GUESS, OK	Manager	
39	giron-d	darron.giron@enron.com	Daron Giron	OK	Employee	
40	griffith-j	john.griffith@enron.com	John Giffith	OK	Managing Director	MD of UK
41	grigsby-m	mike.grigsby@enron.com	Michael Grigsby	OK	Manager	
42	guzman-m	mark.guzman@enron.com	Mark Guzman	OK	Trader	
43	haedicke-m	mark.haedicke@enron.com	Mark Haedicke	OK	Managing Director	Legal Department
44	hain-m	mary.hain@enron.com	Mary Hain	OK	In House Lawyer	
45	harris-s	steven.harris@enron.com	Steven Harris	OK	N/A	
46	hayslett-r	rod.hayslett@enron.com	Rod Hayslett	OK	Vice President	Also Chief Financial Officer and Treasurer
47	heard-m	marie.heard@enron.com	Marie Heard	OK	N/A	
48	hendrickson-s	scott.hendrickson@enron.com	Scott Hendrickson	OK	N/A	
49	hernandez-j	judy.hernandez@enron.com	Juan Hernandez	FAIL judy.hernandez@enron.com, juan.hernandez@enron.com	N/A	
50	hodge-j	t..hodge@enron.com	John Hodge	FAIL jeffrey.hodge@enron.com, **john.hodge@enron.com** and t..hodge@enron.com	Managing Director	
51	holst-k	keith.holst@enron.com	Keith Holst	OK	Director	
52	horton-s	stanley.horton@enron.com	Stanley Horton	OK	President	Enron Gas Pipeline
53	hyatt-k	kevin.hyatt@enron.com	Kevin Hyatt	OK	Director	Pipeline Business
54	hyvl-d	dan.hyvl@enron.com	Dan Hyvl	OK	Employee	
55	jones-t	tana.jones@enron.com	Tana Jones	OK	N/A	
56	kaminski-v	vince.kaminski@enron.com	Vince Kaminski	OK	Manager	Risk Management Head
57	kean-s	steven.kean@enron.com	Steven Kean	OK	Vice President	Vice President & Chief of Staff
58	keavey-p	peter.keavey@enron.com	Peter Keavey	OK	Employee	
59	keiser-k	kam.keiser@enron.com	Kam Keiser	OK	Employee	
60	king-j	jeff.king@enron.com	Jeff King	OK	Manager	
61	kitchen-l	louise.kitchen@enron.com	Louise Kitchen	OK	President	Enron Online
62	kuykendall-t	tori.kuykendall@enron.com	Tori Kuykendall	OK	Trader	
63	lavorato-j	john.lavorato@enron.com	John Lavorato	OK	CEO	Enron America
64	lay-k	elizabeth.lay@enron.com	Kenneth Lay	FAIL elizabeth.lay@enron.com, rosalee.fleming@enron.com and **kenneth.lay@enron.com**	CEO	
65	lenhart-m	matthew.lenhart@enron.com	Matthew Lenhart	OK	Employee	
66	lewis-a	andrew.lewis@enron.com	Andrew Lewis	OK	Director	
67	linder-e	eric.linder@enron.com	Eric Linder	OK	N/A	Specialist
68	lokay-m	michelle.lokay@enron.com	Michelle Lokay	OK	Employee	Administrative Asisstant
69	lokey-t	teb.lokey@enron.com	Teb Lokey	OK	Manager	Regulatory Affairs
70	love-p	phillip.love@enron.com	Phillip Love	OK	N/A	
71	lucci-p	t..lucci@enron.com	Paul T. Lucci	OK	Employee	

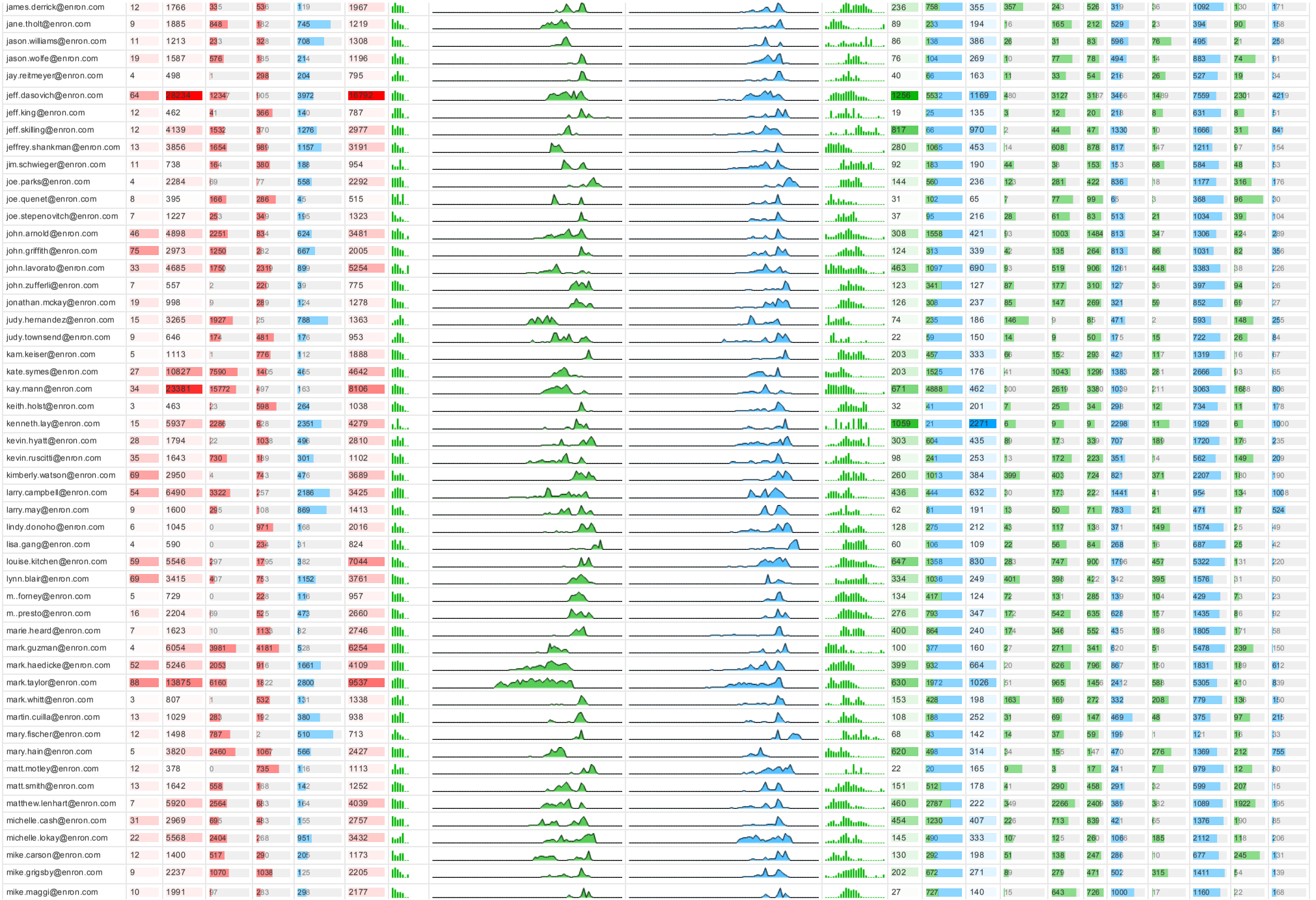
ID	FOLDER NAME	GUESSED EMAIL ADDRESS	FULL NAME	STATUS	TITLE	OTHER INFO
72	maggi-m	mike.maggi@enron.com	Michael Maggi	OK	Director	
73	mann-k	kay.mann@enron.com	Kay Mann	OK	Employee	
74	martin-t	a..martin@enron.com	Thomas A. Martin	OK	Vice President	
75	may-l	larry.may@enron.com	Larry May	OK	Director	
76	mccarty-d	danny.mccarty@enron.com	Danny McCarty	OK	Vice President	
77	mcconnell-m	mike.mcconnell@enron.com	Mike McConnell	OK	N/A	
78	mckay-b	brad.mckay@enron.com	Bradley Mckay	OK	Employee	
79	mckay-j	jonathan.mckay@enron.com	Jonathan Mckay	OK	Director	
80	mclaughlin-e	errol.mclaughlin@enron.com	Errol McLaughlin	OK	Employee	
81	merriss-s	pete.davis@enron.com	Steven Merriss	FAIL (is a GUESS) **steven.merris@enron.com**	N/A	Specialist
82	meyers-a	bert.meyers@enron.com	Albert Meyers	OK	Employee	
83	mims-thurston-p	patrice.mims@enron.com	Patrice Mims-Thurston	OK	N/A	
84	motley-m	matt.motley@enron.com	Matthew Motley	OK	Director	
85	neal-s	scott.neal@enron.com	Scott Neal	OK	Vice President	
86	nemec-g	gerald.nemec@enron.com	Gerald Nemec	OK	N/A	
87	panus-s	stephanie.panus@enron.com	Stephanie Panus	OK	Employee	
88	parks-j	joe.parks@enron.com	Joe Parks	OK	N/A	
89	pereira-s	susan.pereira@enron.com	Susan Pereira	OK	Employee	
90	perlingiere-d	debra.perlingiere@enron.com	Debra Perlingiere	OK	N/A	
149	phanis-s	stephanie.panus@enron.com	Stephanie Panus	DUPLICATE OF panus-s ID 87		
91	pimenov-v	vladi.pimenov@enron.com	Vladi Pimenov	OK	N/A	
92	platter-p	phillip.platter@enron.com	Phillip Platter	OK	Employee	Sr.Specialist
93	presto-k	m..presto@enron.com	Kevin M. Presto	MULTIPLE m..presto@enron.com, kevin.presto@enron.com	Vice President	
94	quenet-j	joe.quenet@enron.com	Joe Quenet	OK	Trader	
95	quigley-d	dutch.quigley@enron.com	Dutch Quigley	OK	N/A	
96	rapp-b	bill.rapp@enron.com	Bill Rapp	OK	N/A	
97	reitmeyer-j	jay.reitmeyer@enron.com	Jay Reitmeyer	OK	Employee	
98	richey-c	cooper.richey@enron.com	Cooper Richey	OK	Manager	
99	ring-a	andrea.ring@enron.com	Andrea Ring	OK	N/A	
100	ring-r	richard.ring@enron.com	Richard Ring	OK	Employee	
101	rodrique-r	robin.rodrigue@enron.com	Robin Rodrigue	GUESS, OK	N/A	
102	rogers-b	benjamin.rogers@enron.com	Benjamin Rogers	OK	N/A	
103	ruscitti-k	kevin.ruscitti@enron.com	Kevin Ruscitti	OK	Trader	
104	sager-e	elizabeth.sager@enron.com	Elizabeth Sager	OK	Employee	
105	saibi-e	eric.saibi@enron.com	Eric Saibi	OK	Trader	
106	salisbury-h	holden.salisbury@enron.com	Holden Salisbury	OK	Employee	Cash Analyst
107	sanchez-m	monique.sanchez@enron.com	Monique Sanchez	OK	N/A	
108	sanders-r	richard.sanders@enron.com	Richard Sanders	OK	Vice President	Enron WholeSale Services
109	scholtes-d	diana.scholtes@enron.com	Diana Scholtes	OK	Trader	
110	schoolcraft-d	darrell.schoolcraft@enron.com	Darrell Schoolcraft	OK	N/A	
111	schwieger-j	jim.schwieger@enron.com	Jim Schwieger	OK	Trader	
112	scott-s	susan.scott@enron.com	Susan Scott	OK	N/A	
113	semperger-c	cara.semperger@enron.com	Cara Semperger	OK	Employee	Senior Analyst Cash
114	shackleton-s	sara.shackleton@enron.com	Sara Shackleton	OK	N/A	
115	shankman-j	jeffrey.shankman@enron.com	Jeffrey Shankman	OK	President	Enron Global Mkts
116	shapiro-r	richard.shapiro@enron.com	Richard Shapiro	OK	Vice President	Regulatory Affairs
117	shively-h	hunter.shively@enron.com	Hunter Shively	OK	Vice President	
118	skilling-j	jeff.skilling@enron.com	Jeffery Skilling	OK	CEO	
119	slinger-r	ryan.slinger@enron.com	Ryan Slinger	OK	Trader	
120	smith-m	matt.smith@enron.com	Matt Smith	OK	N/A	
121	solberg-g	geir.solberg@enron.com	Geir Solberg	OK	Employee	
122	south-s	steven.south@enron.com	Steven P. South	OK	N/A	
123	staab-t	theresa.staab@enron.com	Theresa Staab	OK	Employee	
125	steffes-j	d..steffes@enron.com	James D. Steffes	OK	Vice President	Government Affairs
126	stepenovitch-j	joe.stepenovitch@enron.com	Joe Stepenovitch	OK	Vice President	Enery marketting and trading Florida
127	stokley-c	chris.stokley@enron.com	Chris Stokley	OK	N/A	
128	storey-g	geoff.storey@enron.com	Geoffery Storey	OK	Director	
129	sturm-f	fletcher.sturm@enron.com	Fletcher Sturm	OK	Vice President	
130	swerzbin-m	mike.swerzbin@enron.com	Mike Swerzbin	OK	Trader	
131	symes-k	kate.symes@enron.com	Kate Symes	OK	Employee	
132	taylor-m	mark.taylor@enron.com	Mark Taylor	OK	Employee	
133	tholt-j	jane.tholt@enron.com	Jane Tholt	OK	Vice President	
134	thomas-p	d..thomas@enron.com	Paul D. Thomas	OK	N/A	
135	townsend-j	judy.townsend@enron.com	Judy Townsend	OK	Employee	
136	tycholiz-b	barry.tycholiz@enron.com	Barry Tycholiz	OK	Vice President	
137	ward-k	.ward@enron.com	Kim Ward	MULTIPLE kim.ward@enron.com, .ward@enron.enron.comcom, s..ward@	N/A	
138	watson-k	kimberly.watson@enron.com	Kimberly Watson	OK	N/A	
139	weldon-c	v.weldon@enron.com	Charles V. Weldon	MULTIPLE v.weldon@enron.com and charles.weldon@enron.com	N/A	
140	whalley-g	greg.whalley@enron.com	Lawrence Greg Whalley	OK	President	
150	whalley-l	greg.whalley@enron.com	Greg Whalley	DUPLICATE OF whalley-g ID 140		
141	white-s	w..white@enron.com	Stacey W. White	OK	N/A	
142	whitt-m	mark.whitt@enron.com	Mark Whitt	OK	N/A	
143	williams-j	jason.williams@enron.com	Jason Williams	OK	N/A	Trading
144	williams-w3	bill.williams@enron.com	Bill Williams III	OK	N/A	Senior Analyst

C Output of the dataset overview

Overview of all email accounts

All the email accounts with key statistics. Some columns can be clicked to open more detailed visualizations. These are marked with a darker table header. Change the histogram **type**.

E-mail	Folders	Original messages	Duplicates	Restored	Mismatches	Current messages	Weekday	Histogram sent	Histogram received	TimeOfDay	Unique to	Messages sent	Unique from	Forwarded	Replies	DM Sent	DM Received	Suspects Sent	Suspects Received	External Sent	External Received
.ward@enron.com	49	2611	568	131	963	2174					251	629	337	128	275	529	939	122	760	209	358
a..martin@enron.com	10	1112	111	177	440	1178					91	212	226	34	124	182	338	31	547	32	44
andrea.ring@enron.com	8	706	306	378	109	778					70	186	121	72	53	183	86	55	497	13	16
andrew.lewis@enron.com	10	2191	407	248	1715	2032					27	42	486	9	21	39	1485	2	356	25	962
andy.zipper@enron.com	26	1563	111	679	199	2131					331	412	296	58	225	327	505	87	1526	106	164
barry.tycholiz@enron.com	9	1219	43	815	54	1991					191	575	258	85	246	458	394	218	1365	93	123
benjamin.rogers@enron.com	23	8009	4955	264	574	3318					309	962	439	30	375	786	1227	25	1783	385	411
bert.meyers@enron.com	3	1099	0	1923	1047	3022					28	60	66	6	4	36	126	39	2851	3	9
bill.rapp@enron.com	7	563	6	136	198	693					91	125	199	14	43	56	242	63	370	10	11
bill.williams@enron.com	24	3440	0	2737	665	6177					199	705	276	151	212	384	729	458	4961	61	285
brad.mckay@enron.com	8	681	169	413	278	925					51	102	175	20	74	100	243	7	546	73	134
cara.semperger@enron.com	13	721	0	738	120	1459					188	483	192	55	129	260	319	220	856	99	67
carol.clair@enron.com	14	3030	1506	1541	66	3065					327	1475	182	27	566	840	276	525	1529	430	61
chris.dorland@enron.com	11	2127	820	316	56	1623					249	1025	155	86	506	928	191	95	544	480	86
chris.germany@enron.com	69	12436	6710	334	410	6060					563	3638	437	317	1449	2119	868	569	2016	1108	447
chris.stokley@enron.com	20	1252	0	304	213	1556					209	517	125	167	224	439	263	21	831	87	245
clint.dean@enron.com	11	2429	655	272	1276	2046					43	81	242	21	37	68	610	12	1420	56	225
cooper.richey@enron.com	22	605	0	159	77	764					112	383	168	20	231	347	155	7	305	226	62
d..steffes@enron.com	64	3331	57	1012	589	4286					402	1459	422	404	794	741	625	395	2308	131	311
d..thomas@enron.com	6	1293	2	128	457	1419					68	159	282	29	75	143	813	8	806	71	333
dan.hyvl@enron.com	45	3210	1578	400	233	2032					283	693	265	56	408	439	503	152	1110	138	151
dana.davis@enron.com	33	2249	1118	918	246	2049					146	361	361	70	121	285	430	48	1442	205	123
danny.mccarty@enron.com	23	691	1	444	143	1134					118	200	229	60	71	164	169	61	793	19	19
daren.farmer@enron.com	40	13032	7799	140	2275	5373					199	753	585	18	524	602	1397	7	2456	64	1120
darrell.schoolcraft@enron.com	18	1859	81	417	453	2195					254	507	234	265	144	304	355	88	1251	22	94
darron.giron@enron.com	11	4220	2266	202	1095	2156					206	818	259	143	235	630	630	119	444	322	287
david.delainey@enron.com	8	3566	2561	1110	201	2115					328	810	256	17	229	455	171	264	1176	11	131
debra.perlingiere@enron.com	11	4778	1847	756	213	3687					531	2447	220	120	1039	2047	340	395	1028	731	45
diana.scholtes@enron.com	26	647	0	950	122	1597					118	301	176	124	98	207	169	117	1177	72	161
don.baughman@enron.com	57	2760	953	353	327	2160					227	208	445	63	44	87	790	76	1628	63	257
doug.gilbert-smith@enron.com	21	578	0	754	192	1332					57	118	287	22	49	82	206	63	1023	18	150
drew.fossum@enron.com	7	4796	3246	784	172	2334					352	1177	239	90	615	613	147	406	1008	120	56
dutch.quigley@enron.com	16	1568	68	418	356	1918					160	542	270	40	338	469	638	83	1023	174	213
elizabeth.sager@enron.com	14	5200	1944	1284	303	4540					393	1497	495	155	705	1197	620	203	2750	434	384
eric.bass@enron.com	15	7823	4724	433	751	3532					235	1711	335	147	918	1289	975	323	1074	594	457
eric.linder@enron.com	4	2805	1859	838	192	1784					22	13	69	2	3	9	154	7	1606	0	4
eric.saibi@enron.com	8	1116	25	271	661	1362					33	40	191	7	10	36	467	7	667	18	413
errol.mclaughlin@enron.com	20	3353	1747	400	360	2006					160	511	283	46	220	357	563	171	1141	25	92
frank.ermis@enron.com	8	1230	517	245	406	958					22	30	189	1	9	27	385	15	543	6	122
geir.solberg@enron.com	6	1081	0	4042	123	5123					52	88	91	8	9	73	107	40	4928	17	10
geoff.storey@enron.com	19	1027	150	475	306	1352					68	154	271	50	60	141	393	21	897	87	109
gerald.nemec@enron.com	12	10655	4220	296	597	6731					590	2231	632	213	689	1500	1907	407	3917	691	504
greg.whalley@enron.com	32	5213	3281	1379	792	3311					121	188	601	28	112	159	606	53	2387	20	211
harry.arora@enron.com	14	654	162	325	122	817					57	87	251	21	47	79	279	17	610	20	106
holden.salisbury@enron.com	5	1632	0	701	941	2333					92	155	225	20	59	117	780	29	1341	26	108
hunter.shively@enron.com	15	1991	771	480	898	1700					111	305	275	6	123	279	427	45	833	20	80
j..sturm@enron.com	21	1169	419	324	303	1074					94	166	177	29	74	152	334	25	668	21	27



mike.mcconnell@enron.com	79	4542	2700	991	759	2833					425	789	495	84	453	628	447	171	1492	115	173
mike.swerzbin@enron.com	5	355	0	891	123	1246					47	75	179	27	19	59	147	12	1051	16	78
monika.causholli@enron.com	4	943	9	1498	148	2432					131	501	141	24	160	359	260	17	1783	58	72
monique.sanchez@enron.com	7	256	8	313	80	561					110	118	95	24	51	98	160	35	363	77	30
patrice.mims@enron.com	8	2038	557	234	1229	1715					87	245	366	2	159	232	950	18	362	143	151
paul.y'barbo@enron.com	20	1291	204	357	232	1444					118	192	206	32	67	110	210	25	1020	31	125
peter.keavey@enron.com	10	2177	890	120	1147	1407					48	74	207	0	24	63	957	17	280	21	567
phillip.allen@enron.com	10	3034	1637	918	767	2315					201	619	310	10	268	547	610	145	1067	262	441
phillip.love@enron.com	21	5002	2502	332	1484	2832					297	900	304	104	490	698	972	112	806	263	235
phillip.platter@enron.com	6	574	0	675	272	1249					77	119	136	45	39	86	242	8	861	51	121
richard.ring@enron.com	14	994	11	158	345	1141					72	108	259	31	46	84	435	9	690	20	288
richard.sanders@enron.com	50	7329	2990	942	1969	5281					627	1657	574	97	987	1201	1052	204	1899	526	1229
richard.shapiro@enron.com	116	6071	2700	4134	718	7505					443	519	670	101	224	302	1249	322	6288	72	1396
rick.buy@enron.com	15	2429	522	732	261	2639					364	585	501	104	314	427	400	64	1800	69	159
rob.gay@enron.com	9	1415	842	11	345	584					129	213	84	2	121	121	227	2	109	69	54
robert.badeer@enron.com	16	877	553	1204	160	1528					50	89	180	5	29	55	129	13	1291	16	357
robert.benson@enron.com	10	767	167	276	277	876					17	23	159	1	13	23	168	0	579	7	43
robin.rodrigue@enron.com	8	2766	1943	183	3	1006					148	742	82	19	429	615	58	38	261	58	3
rod.hayslett@enron.com	49	2554	784	855	154	2625					388	783	460	247	436	419	486	339	1692	41	61
ryan.slinger@enron.com	3	132	0	4868	38	5000					42	81	73	10	8	59	70	45	4885	7	10
sally.beck@enron.com	135	11830	6897	815	569	5748					1294	1572	716	164	716	1139	1098	186	3626	156	90
sandra.brawner@enron.com	13	1026	424	269	409	871					43	162	147	60	105	155	185	13	452	116	41
sara.shackleton@enron.com	111	18687	8059	1214	1015	11842					756	4447	837	329	1464	3123	2894	1136	6406	742	851
scott.hendrickson@enron.com	9	719	193	268	218	794					54	68	173	6	30	52	375	8	508	50	133
scott.neal@enron.com	21	3268	1492	708	355	2484					305	658	392	79	323	524	478	138	1475	154	188
sean.crandall@enron.com	11	519	0	892	142	1411					78	169	194	14	85	102	216	35	1101	47	115
shelley.corman@enron.com	24	2025	44	1333	560	3314					513	739	474	133	347	490	765	183	2026	215	303
stacy.dickson@enron.com	3	395	196	972	10	1171					72	221	99	20	108	167	88	55	940	38	36
stanley.horton@enron.com	9	2470	1141	618	284	1947					292	467	400	28	376	377	323	84	1219	171	237
stephanie.panus@enron.com	9	472	3	1854	57	2323					422	400	205	55	78	140	235	279	1866	59	62
steven.harris@enron.com	2	548	0	1884	214	2432					90	152	340	22	101	99	327	120	2068	20	235
steven.kean@enron.com	193	25351	19116	1353	2660	7588					434	1412	828	150	559	900	1364	289	4389	128	1274
steven.merris@enron.com	4	1627	1076	72	182	623					4	4	67	0	1	4	147	2	465	2	5
steven.south@enron.com	7	248	130	150	78	268					11	21	68	3	11	21	52	0	189	13	30
susan.bailey@enron.com	5	478	17	2162	70	2623					348	238	202	25	79	124	312	125	2318	38	62
susan.pereira@enron.com	9	725	312	199	284	612					39	126	92	6	89	122	139	6	281	89	82
susan.scott@enron.com	21	8022	4468	942	2032	4496					739	1186	594	66	501	824	1714	316	1502	351	1109
t..hodge@enron.com	10	1661	220	412	990	1853					85	122	388	37	33	86	603	54	969	11	223
t..lucci@enron.com	5	997	9	89	478	1077					100	295	177	72	181	235	535	51	306	194	147
tana.jones@enron.com	10	19950	9281	679	673	11348					882	4165	766	220	1632	2285	2737	1363	6527	373	416
teb.lokey@enron.com	7	1156	0	225	523	1381					138	138	249	42	77	77	458	29	722	12	256
theresa.staab@enron.com	17	621	19	157	236	759					83	157	155	20	68	87	287	57	367	45	49
tom.donohoe@enron.com	8	1015	328	123	403	810					37	44	157	9	13	38	250	7	401	24	60
tori.kuykendall@enron.com	9	1120	542	319	115	897					153	320	150	56	170	271	151	33	463	202	65
tracy.geaccone@enron.com	18	1592	16	315	137	1891					246	668	268	197	343	453	418	213	1088	51	26
v.weldon@enron.com	41	1566	589	86	574	1023					123	280	202	11	153	241	362	16	228	106	86
vince.kaminski@enron.com	54	28465	16909	224	5284	11780					856	3992	1521	136	1861	2351	4361	1476	2599	1945	2201
vladi.pimenov@enron.com	8	642	3	178	286	817					42	78	152	9	40	72	255	7	455	21	149
w..white@enron.com	34	3272	397	235	1036	3110					184	486	309	129	202	308	444	63	1635	28	119

D Implementation details

D.1 Parsing PST and MSG files

A PST wrapper named *parse_pst.py* run *readpst* on every *PST* file in the project *pst* folder. It keeps the MSG files and unlinks EML files. A MSG parser named *parse_msg.py* then use the *OleFileIO_PL*[84] library for parsing these structured compound files.

The MSG parser is inspired by the *msg-extractor*[93] program. It rebuilds a multi-part email message based on the original header field. The Python *email library* is used for assembly of the message. Any HTML, plain text and attachment found are appended to the new message. Attachments are re-encoded using *base64*. Some minor modifications to the original content-type were needed since some of the original messages did not contain multiple parts. This header would otherwise conflict with the newly generated multi-part *content-type*.

The source *MSG* files are unlinked after the new *EML* file is saved to disk. This is done for space conservation.

D.2 Meta data extraction: Parsing EML files

metadata.py

The *metadata.py* program traverse the folder structure of an account, reads the files one by one and constructs an email object from them using the built in Python *email* library. A Python list is filled with an entry for every email message consisting of the collected and processed fields.

Date is given as a string in the message. The function *email.utils.parsedate* can return a time object that can be turned into a UNIX timestamp. Unix timestamps are measured in seconds since the start of the epoch, defined as 1st of January 1970. Timestamps are useful because two timestamps can be directly compared in terms of how much time has passed using subtraction. The time string is in local time and has a time zone parameter. It is not handled by the *parsedate* function. A regular expression match is used to find the offset. It is added to the local timestamp to calculate the *UTC* time which is the timezone crossing Greenwich at +0 hours. Local time of day and day of week is also added at this point. Timezone offset can be up to 12 hours plus or minus.

Subject is decoded using the *email.header.decode_header* function and converted to unicode. The length is determined and it is searched for the configured regex forward and reply strings.

All email address fields are split by *;* or *,* and parsed by the *email.utils.parseaddr* function. The result of this function is pairs of *realname* and *email address*. Only the address is used when *To*, *Cc* and *Bcc* are combined into a generic *receivers* list. The same applies for the *From* field, but it is stored as a string since the norm is one sender. Number of internal and external receivers are counted based on domain specification in the *settings.py* file. A boolean flag *draft* is set when either receivers or sender is missing.

The original headers are decoded using *email.parser.HeaderParser().parsestr()*. Header keys

and values are split up and potentially dangerous characters in the values are replaced safe substitutes. The user id and name of the original *EML* file is added to the headers as *meta_data_source*. HTML content and plain text content is also fetched and stored in a JSON file. This file is loaded dynamically when a user clicks a message in the *message content view*. This preprocessing consumes time and could be done on the fly. Attachments are decoded and stored with the MD5 sum of the content as the filename.

A few other measures are done, and they are described in the following Table 8.

ID	Field name	Description	Type of data
0	meta_username	Tracking meta data (context)	String
1	meta_folder	Tracking meta data	String
2	meta_messageid	Tracking meta data	String
3	timestamp_local	Seconds since 1970	Integer
4	timestamp_utc	Adjusted to UTC timezone based on timezone	Integer
5	timezone	Number of hours offset	Integer [-12..+12]
6	local_weekday	Local day of week	Integer [0..6]
7	local_hour	Local hour of day	Integer [0..23]
8	email_subject	The decoded subject line	String
9	email_subject_length	The length of the subject line	Integer
10	email_subject_forward	If subject contains "fw:" or "fwd:"	Boolean
11	email_subject_reply	If subject contain "re:"	Boolean
12	internal_receivers	Number of addresses receiver list not to the specified company domain	Integer
13	external_receivers	Receivers within the company domain	Integer
14	email_sender	The decoded email address in the "From" field	String
15	email_receivers	The decoded email addresses in the To, Cc and Bcc fields	List
16	draft	if no sender address or no receivers	Boolean
17	email_payload_len	Length of either HTML, plain text part or the entire message in that order	Integer
18	parts_multipart	Number of attachments	Integer
19	parts_plain	Number of HTML parts	Integer
20	parts_html	Number of plain text parts	Integer
21	parts_other	Number of other things attached, if any	Integer
22	email_folders	All folders the message has been seen in. Filled when removing duplicates.	List
23	attachments	List of files with triples of filename, md5 sum and content tpe for each file	List
24	content_id	The ID of the JSON file with HTML and plain content. Used for dynamic loading.	Integer

Table 8: Description of extracted meta data for *metadata.py*

Removal of duplicate messages can start once all messages has been read and added. The list of message meta data is sorted by UTC time as primary index and content length as secondary index. The check for duplicates is simplified to looping through the messages and comparing the current to the last message. Two messages with the same timestamp can be duplicates, and sorting by content size ensures that duplicate candidates follow each other even if they have the same timestamp. A match is declared if meta data ranging from number 5 including 16 in Table 8 match exactly. The *email_folders* attribute is originally set to the folder where the message was found. This field is updated with the folders of the discarded duplicate messages.

email_sender and *email_receivers* now contain information about who is communicating. Messages in a user folder are either incoming messages or local copies of outgoing mail. This means that the *email_receivers* field will not always be "to the current user". The technique used is first to gain knowledge of the current users email address and then use this knowledge to determine what direction the message was sent.

All unique email addresses in the *email_sender* field of the messages are counted after duplicates have been removed. Usernames in the Enron dataset consist of last name followed by a dash and the first letter in the given name. This knowledge together with company domain is

used to make a regular expression search starting from the most common address and moving down. If no match can be found then the most frequent address is guessed to be the correct one. This procedure is simplified when *PST* files are used as input since the *maildir* folders are given the name of the email account found inside by *readpst*.

The username, determined email address, number of original and duplicate messages are written to a JSON file together with attribute descriptions and the data list. All pairs of username and email address are written to an address book when all accounts have been parsed. Aggregated statistics are displayed in the console including runtime.

direction.py

This program uses the knowledge of the owners address and sets the *sent flag* if the address is found in the *email_sender* field. A flag *other_party_suspect* is set if the other party in the conversation is among the set of collected accounts. These messages are also collected in a list of candidates for recovery. The last thing done for each meta data file is to add a flag *email_missing* if the current users address was neither found in the sender or receivers field.

New attributes are shown in Table 9:

ID	Field name	Description	Data type
25	email_sent	True if sent by user	Boolean
26	other_party_suspect	True if another collected account was involved	Boolean
27	email_missing	True if users email address was not present	Boolean

Table 9: Description of extracted meta data for direction.py

All candidates for recovery are cleaned by removing duplicates using the same principle as described previously. The remaining messages are saved to a file called *candidate_messages.json* in the *processed* folder. Removal of duplicates is importance since a message can be found in many accounts and this will reduce the computational complexity of adding messages back later.

undelete.py

For each message in the *candidate message list* generated by the previous program, look at the *receivers* field if message was sent. If the message was received look at both *sender* and *receivers*. If the current users email address is found amount the addresses then a list of all messages belonging to the current user with the same timestamp as the candidate message is constructed.

The candidate message is added to the users list of messages if *subject* and *message length* are not seen in among messages with the same timestamp. The *direction flag* is inverted in order to reflect the new point of view and a new flag *added_back* is set to true. Old messages get this flag set to false. The personal folder structure details is emptied since this information is no longer valid. The *mismatch flag* is also reset as it is not possible to be mismatch when it has been recovered.

New attributes are shown in Table 10:

ID	Field name	Description	Type of data
28	added_back	True if put back from another users account. Implies that other_party_suspect is also set	Boolean

Table 10: Description of extracted meta data for undelete.py

timestatistics.py

This program calculates the number of messages in the direction of each message for the current day and the current week. This is done using counters during a first pass through the data where unique dates are counted. These counters are looked up during the 2nd pass and appended to the meta data vector. Unique people per day is also gathered by aggregating daily sums up of the unique *sender* fields if received and the unique *receivers* is sent. This gives the messages time context data.

New attributes are shown in Table 11:

ID	Field name	Description	Type of data
29	messages_day	Number of messages in messages direction that day	Integer
30	messages_week	Number of messages in messages direction that week	Integer
31	unique_people_day	Number of unique people received from or sent to that day, depending on direction	Integer

Table 11: Description of extracted meta data for timestatistics.py

recover.py

This program reads in a user supplied email address. I will scan through all the existing account meta data files and look for the specified address, but only if the address does not exits in the *address book*. Logic for what fields to look in is the same as for *undelete.py* and discovered messages are written to a new meta data file once duplicates are removed. The address book is updated and the *timestatistics.py* program is targeted at this new meta data file.

Gmail is one example of an email on-line service that does not care about dots (.) and everything after plus sign (+)[1] in email addresses. This is the explanation for the *simplity_address(address)* function.

D.3 The django module

All files related to the *Django* module is inside the *server* folder except for *manage.py* which is used for managing the Django project. A development server can be started by issuing the command "*python manage.py runserver 0.0.0.0:8000*" from the folder where this file is located, usually the folder containing the *server* folder A Production environment would require integration with a traditional server such as Apache[2] or Nginx[3] as a module using the web server gateway interface.

In order to explain how an HTTP requests ends up with a HTML response, see Figure 65.

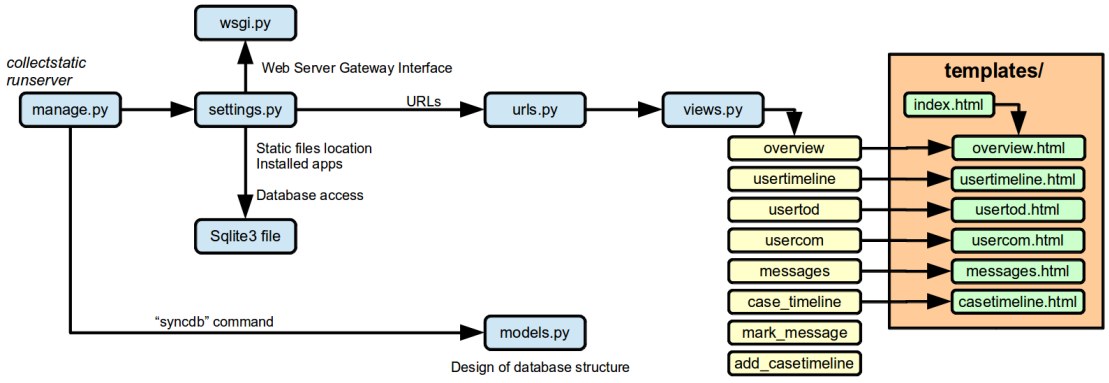


Figure 65: An overview of the most important files in the Django project.
(Simplified overview, not a true representation of internal workings of Django (add collect static and runserver to ill))

The file *manage.py* refers *server/settings.py* mentioned earlier. It stores information about how to access databases, what applications should be loaded and what file specifies how HTTP requests should be handled. This file is by default *urls.py* and it uses regular expressions for request matching. See Figure 66 for an example:

[1] 2 hidden ways to get more from your Gmail address: `http://gmailblog.blogspot.no/2008/03/2-hidden-ways-to-get-more-from-your.html`

[2] `http://www.apache.org/`

[3] `http://nginx.org/`

```
from django.conf.urls import patterns, url

urlpatterns = patterns(
    '',
    url(r'^messages/(?P<user_id>[^/]+)/$', 'server.views.messages', name='messages'),
)
```

Match regex — Handling function

Figure 66: This regular expression matches a line starting with "messages/" and ending with "/". Everything between is assigned to the variable user_id and passed on to the function messages in the views.py file (in the server folder).

One of the major benefits of Django is the template engine. Functions defined in *views.py* gather data and supply them to a template where the values are formatted and inserted into the HTML structure. All templates inherit and append to the *index.html* file. Figure 65 shows all the functions in *views.py*. Some of them import functionality from other files in order to keep *views.py* clean.

Static files such as images, JavaScript libraries and Cascading Style Sheets (CSS) are normally served outside of the scope of *Django*. This is why *settings.py* has the variables *STATICFILES_DIRS* and *STATIC_URL*. They are used for synchronizing static files in a Django application folder with the absolute path to the folder where the web server serves them from. The "*python manage.py collectstatic*" is used for the synchronization.

A simple SQLite3 database is being used to store what messages has been read and what messages has been added to the *case timeline*. This database file is stored together with the meta data files. Database design is done in *models.py* and set up using the "*python manage.py syncdb*" command.

The following sub chapters will explain the *Django views* in *views.py*:

overview(): Dataset overview

This view imports functions from *overview.py* and outputs via *overview.html*. It uses multiple processes in order to read and assemble statistics from all available accounts. It is the front page and index of the web service. Table 12 summarizes what is collected:

Count per account of	How
Number of user folders	Gathered while parsing
Messages originally	Gathered while parsing
Messages in total, sent and received	Using the direction flag
Duplicate messages	Gathered while parsing
Messages recovered	Gathered while parsing
Direct messages sent and received	If number of recipients is 1
Forwarded messages (sent)	Using the forward flag
Replied messages (sent)	Using the reply flag
Sent to and received from external domains	If number of external addresses is 1 or more
Unique addresses sent to and received from	Check every address, add it if not already existing

Table 12: Data counted and collected for each user (new forgotten?)

The function generate the data necessary for plotting histograms of the sent *hour of day* and *day of week*. Weekly aggregate of both sent and received messages are also created. *overview.html*

creates a table with one user per row. *jQuery.floatThead*[94] is used for making sure the table header is always visible.

Weekday, *histograms* and *TimeOfDay* make use of the jQuery Sparkline [95] library for plotting. Some preprocessing is done for the histogram data in order to fill non-existing weeks.

messages(): Message content view

This view imports functions from *messages.py* and outputs via *messages.html*. The data supplied to the template in the raw message view is a subset of all the data in the JSON meta data file for a selected user account. The rationale is to limit the amount of data sent between the server and the client. Instead of sending all recipients of a message, only the first is used. A count of how many other addresses was present is also supplied.

The template generates a table for all the messages line by line and it has an area for loading in preview of selected messages.

case_timeline(): The case timeline

An entry for the case timeline is the *user id*, *file id*, *comment* and *timestamp*. The comment is set to the message subject by default and the timestamp is used to draw a timeline using the *Chap links library timeline*[96] library. It is used in off-line JSON mode.

The database only keeps track of the *user id* and the *JSON file id*. These two parameters uniquely identifies any message in the system.

usertimeline(): Weekly timeline

This view imports functions from *usertimeline.py* and outputs via *usertimeline.html*. The function processes a single users meta data file. It counts number of messages and unique addresses in each direction on a daily basis. Data in the sent and received directions are plotted in separate graphs. It was therefore made sure they contained the same range on for time values on the x-axis. These daily counts are returned in series, one for each week day where time between values is one week.

Data Driven Documents *d3*[97] is a library for binding data such as numbers to elements in web pages. It is low level, but has many helper functions for normalization of data, generation of axis and handling of time and date. It is written in JavaScript and is similar to *jQuery*[4] in terms of selecting elements. *nvd3*[98] is a visualization library build on top of *d3*. The purpose is highly customizable, reusable charts while avoiding having to start from scratch. Version 1.1 of *nvd3* is used.

The *nvd3 multiBarChart* chart is used for graphing the data. The visualization has support for transition between stacked and grouped bars. The daily series can be be stacked in order to show weekly counts instead of daily. This functionality is enabled for the message count, but disabled for the unique addresses count. Stacked unique peers per day would probably contain overlapping addresses and thus give a too optimistic sum.

One interesting property of the visualization library used for this graph is that time is not drawn linear. A timestamp of 1980 is right next to 1999 if there are no values dated in between.

[4] jQuery: `http://jquery.com/`

This is practical in terms of not having large gaps in the data but at the same time it makes it hard to consider periods of duration. This behavior can be changed to linear using the address bar variable *linear*. The time period can also be limited using the address bar variable *filter*, *begin* and *end*. See Figure 67.

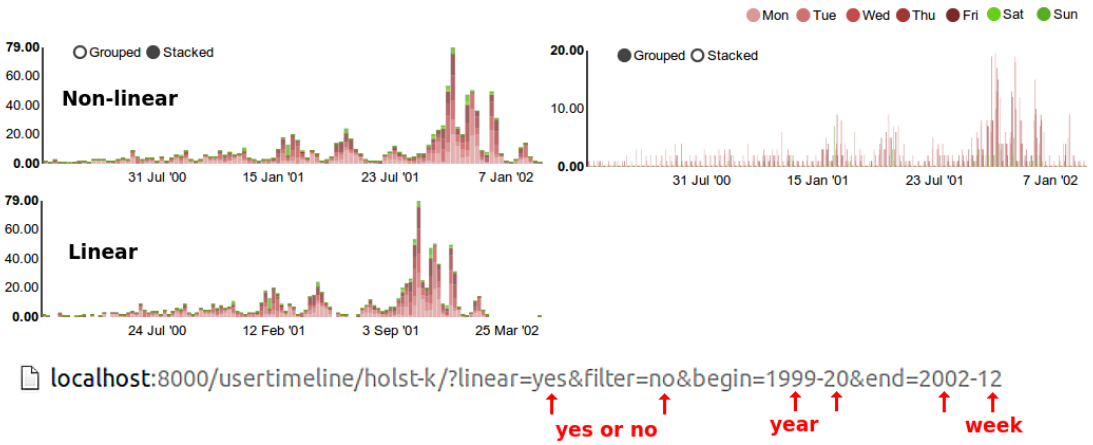


Figure 67: Switching between stack and grouping. Change to linear mode and time filtering. Clicking on day in the graph redirects the browser to the *message content view* set to the selected period.

usertod(): Time of day plot

This view imports functions from *usertod.py* and outputs via *usertod.html*. This function enumerates all the sent message's local day and time. The generated output is shown in Figure 68:

```
// test data: uncomment in order to replace data for verification
newData = [
    {"day": 1262304000, "times": [7,8,9,15,16,17,21,22,23,24], "weekday": 1},
    {"day": 1262400400, "times": [0,1,2,7,8,9,12], "weekday": 2},
    {"day": 1262500400, "times": [8,9,14,15], "weekday": 3},
    {"day": 1262600400, "times": [12,13], "weekday": 4},
    {"day": 1262700400, "times": [12], "weekday": 5},
    {"day": 1262800400, "times": [12,13,17,18,19,23,24], "weekday": 6},
    {"day": 1263304000, "times": [0,1,5,6,12,13,14,19,23,24], "weekday": 1},
//times come in resolution of minutes with 2 decimal places i.e 3.50 is 3 hours and 30 minutes after midnight.
]
```

Figure 68: Data format for visualizing time of day: A list of days. Every day has a *day* timestamp representing the start of the day, a list of decimal *times* where a message was sent (between 0.0 and 24.0) and a *weekday* (0-6) representing Monday to Sunday.

This data is transformed using a custom made d3 visualization. SVG elements are Scalable Vector Graphics[5]. All unique days are linked to SVG rectangles that span from the earliest to the latest time of day. An additional rectangle is added in order to mark the period from 08:00 to

[5] `http://www.w3.org/Graphics/SVG/`

16:00, the typical normal working hours. Individual times for all days then gets linked to SVG circles and plotted like a scatter plot.

The visualization has controls for hiding or showing both lines and circles. Time can be based on timestamps or events. Missing days will not occupy any space on the x-axis when days are plotted by event instead of timestamp. See Figure 69 for an example:

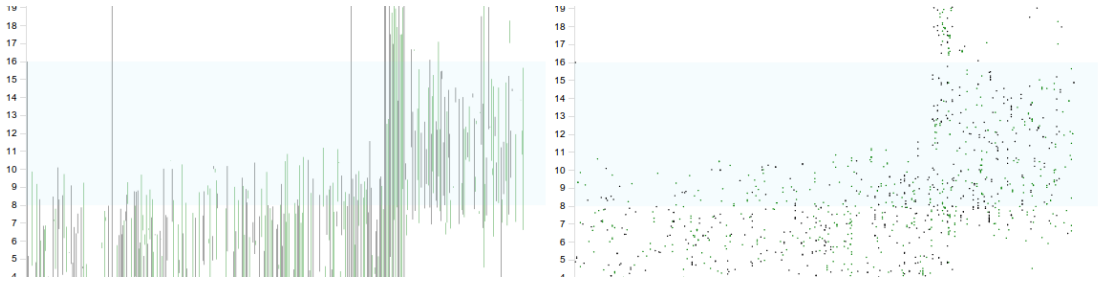
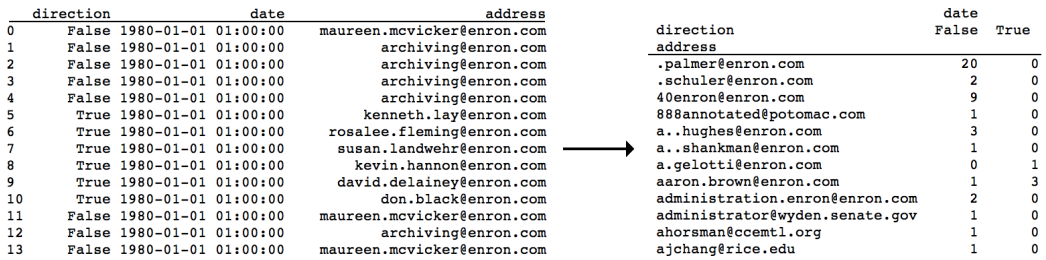


Figure 69: Dots and lines can be turned on and off individually. Clicking a dot will direct to the *message content view* for that particular time.

usercom(): Communication breakdown

This view imports functions from *usercom.py* and outputs via *usercom.html*. *Pandas* is a Python library for adding data manipulations such as indexing, sorting, selections, groupings and join operations similar to SQL[6] queries in relational databases.

This function collects all the *recipients* if sent and the *sender* if received. The resulting data structure is an entry for every address found, its direction and timestamp. This structure is turned into a *Pandas DataFrame*, grouped by address and direction in count mode. This grouping is unstacked and empty values filled with the count of 0. The result of these operations is a list of addresses with counts for each direction as shown in Figure 70:



	direction	date	address
0	False	1980-01-01 01:00:00	maureen.mcvicker@enron.com
1	False	1980-01-01 01:00:00	archiving@enron.com
2	False	1980-01-01 01:00:00	archiving@enron.com
3	False	1980-01-01 01:00:00	archiving@enron.com
4	False	1980-01-01 01:00:00	archiving@enron.com
5	True	1980-01-01 01:00:00	kenneth.lay@enron.com
6	True	1980-01-01 01:00:00	rosalee.fleming@enron.com
7	True	1980-01-01 01:00:00	susan.landwehr@enron.com
8	True	1980-01-01 01:00:00	kevin.hannon@enron.com
9	True	1980-01-01 01:00:00	david.delainey@enron.com
10	True	1980-01-01 01:00:00	don.black@enron.com
11	False	1980-01-01 01:00:00	maureen.mcvicker@enron.com
12	False	1980-01-01 01:00:00	archiving@enron.com
13	False	1980-01-01 01:00:00	maureen.mcvicker@enron.com

	date	
direction	False	True
address		
.palmer@enron.com	20	0
.schuler@enron.com	2	0
40enron@enron.com	9	0
888annotated@potomac.com	1	0
a..hughes@enron.com	3	0
a..shankman@enron.com	1	0
a.gelotti@enron.com	0	1
aaron.brown@enron.com	1	3
administration.enron@enron.com	2	0
administrator@wyden.senate.gov	1	0
ahorsman@ccemtl.org	1	0
ajchang@rice.edu	1	0

Figure 70: The Pandas library significantly reduces the implementation workload.

This list of address counts can be sorted on the number of messages sent or received. This behavior is controlled by a direction variable. Two links from the *dataset overview* link the the

[6]SQL is Structured Query Language. A language for selecting, filtering, grouping and merging data tables

two possibilities. Lines with no address in the chosen sort order are discarded.

Every remaining address is checked for being internal or external. The Pandas library is used for filtering out individual address. These datasets are aggregated by counting number of messages per week in each direction. The final result delivered for graphing consists of two lists. One for internal and one external addresses. Both lists contain all the addresses sorted by the preferred direction. Each address has a count of sent and received messages together with two weekly counts lists with histogram data. See Figure 71 for visual representation:


```
{
"internal":
    [{
    "address": "email address"}
    "received": 21,
    "sent": 19,
        "dates": {
            "received": {
                "2002-5": 3, "2002-4": 3, "2002-6": 1, "2002-1": 1,"2002-2": 4, "2001-42": 3, "2001-46": 3, "2001-48": 2, "2001-39": 1
            },
            "sent": {}
    }, {
    "address": "email address"}
    "received": 33,
    "sent": 12,
        "dates": {
            "received": {},
            "sent": {}
    }],
"external":
    [{
    "address": "email address"}
```


Figure 71: The JSON data prepared for visualization

Javascript code in the associated template uses the jQuery Sparkline API to draw two time series graphs per address, one for sent and one for received mirrored on top of each other.

The visualization has two modes. Sorted by messages sent or messages received. Controls allow for switching between normal and binary plot and a time filter can be applied as shown in Figure 72.

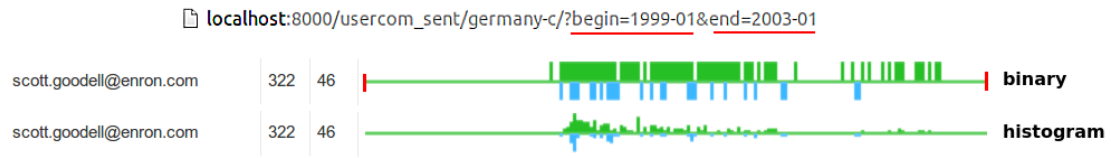


Figure 72: Clicking on any graph directs to the *message content view* with the *address filter* set to the corresponding email address.

E Source code

E.1 runall.py

```
import os
### Parse PST files
os.system('python parse_pst.py')

### Extract meta data from maildir format
os.system('python metadata.py')
os.system('python direction.py')
os.system('python undelete.py')
os.system('python timestatistics.py')

### Start the webserver
os.system('python manage.py syncdb')
os.system('python manage.py runserver 0.0.0.0:8000')
```

E.2 parse_pst.py

```
#This program will run a readpst command on each of the PST files in a folder,
#run the parse_msg.py script on each of them and clean up files.

import os
import timeit  # benchmark time
from parse_msg import parse_msg

from server.settings import global_program_settings as global_settings
PST_PATH = global_settings["pst_path"]
MAILDIR_PATH = global_settings["maildir_path"]

if not os.path.exists(MAILDIR_PATH):
    os.makedirs(MAILDIR_PATH)


def worker(pst_file_path):
    command = ('readpst -D -m -o "%s" "%s"') % (MAILDIR_PATH, pst_file_path)
    # -D    : Include deleted items in the output.
    # -o    : Specifies the output directory
    # -m    : Same as the e option, but write .msg files also
    # -e    : Same as the M option, but each output file will include an extension from (.eml, .ics, .vcf). This format has no from quoting
        .
    # -M    : Output messages in MH (rfc822) format as separate files. This will create folders as named in the PST file, and will put each
        email together with any attachments into its own file. These files will be numbered from 1 to n with no leading zeros. This
        format has no from quoting.
    print("* Running command <%s>") % command
    os.system(command)

    # Remove everything except *.msg files
    print("* Removing all files except *.msg")
    for (dirpath, dirnames, filenames) in os.walk(MAILDIR_PATH):
        for filename in filenames:
            try:
                extention = filename.split(".")[1]
            except:
                extention = None
            if extention != "msg":
                full_path = ("%s/%s") % (dirpath, filename)
                os.remove(full_path)

    # Run the msg to eml program
    print("* Converting .msg to .eml for each user")
    usernames = []
    for (dirpath, dirnames, filenames) in os.walk(MAILDIR_PATH):
        usernames.extend(dirnames)
        break

    for username in usernames:
        subfolder_path = ("%s%s") % (MAILDIR_PATH, username)
        for (dirpath, dirnames, filenames) in os.walk(subfolder_path):
            for filename in filenames:
                if filename.split(".")[-1] == "msg":
                    try:
                        parse_msg(dirpath, filename)
                    except:
                        print("********* %s/%s") % (dirpath, filename)
                        import traceback
                        print traceback.print_exc()


if __name__ == '__main__':
```

```
    timer_start = timeit.default_timer()

    pst_files = []
    for (dirpath, dirnames, filenames) in os.walk(PST_PATH):
        for filename in filenames:
            if filename.split(".")[-1] == "pst":
                pst_files.append(filename)
        break  # do not continue walking
    print("* Found %s Outlook *.pst files") % (len(pst_files))

    for pst_file in pst_files:
        pst_file_path = "%s%s" % (PST_PATH, pst_file)
        worker(pst_file_path)

    timer_stop = timeit.default_timer()
    print("** Completed in %ss") % round(timer_stop - timer_start, 2)
```

E.3 parse_msg.py

```
"""
This program will extract original header, html, plain text and attachments from a MSG file and write back an EML file.
It requires the OleFileIO_PL library (https://pypi.python.org/pypi/OleFileIO_PL)

Sources:
* ExtractMsg.py
* http://stackoverflow.com/questions/3362600/how-to-send-email-attachments-with-python
* http://stackoverflow.com/questions/882712/sending-html-email-using-python
* http://docs.python.org/2/library/email-examples.html
* http://docs.python.org/2/library/email.message.html#email.message.Message.attach

Bugs:
* Read and write in bursts occur, can be very slow.
* Determination of string encoding can be improved
* There is a problem with multiple of the same attachment where one copy is corrupt
"""

from OleFileIO_PL import OleFileIO
from email import message_from_string
from email.Encoders import encode_base64
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.MIMEBase import MIMEBase
import os

FIELD_DESIGNATOR = '__substg1.0_'
ATTACHMENT_DESIGNATOR = '__attach'
EMAIL_TYPE = 'IPM.Note'
MEETING_REQUEST = "IPM.Schedule.Meeting.Request"
DEFAULT_CHARSET = "iso-8859-1"


def invert_dict(dict):
    d = {}
    for a, b in dict.iteritems():
        d[b] = a
    return d

fields = {
    "message_type": "001A",  # should me IPM.Note for email (IPM.Contact, IPM.Post, IPM.Activity, IPM.Task)
    "original_headers": "007D",
    "content_plain": "1000",
    "content_html": "1013",
    "attach_data": "3701",
    "attach_ext": "3703",
    "attach_name_short": "3704",
    "attach_name_long": "3707",
    "attach_mime": "370E",
    "attach_id": "3712",
}
encoding = {
    "UNICODE": "001F",
    "ASCII": "001E",
    "BIN": "0102",
}
fields_inv = invert_dict(fields)  # used for reverse lookup
encoding_inv = invert_dict(encoding)


class Attachment:
    """ Object to store the meta data and content of attachments. """
    def __init__(self, m, directory):
        self.data = m.readStream([directory, fields['attach_data']])
        self.name_long = m.readStream([directory, fields['attach_name_long']])
        self.mime = m.readStream([directory, fields['attach_mime']])

        if "utf-8"" in self.name_long:
            self.name_long = self.name_long[7:]  # remove prepended string


class Message(OleFileIO):
    """ OLE Object with methods for reading internal streams. """
    def __init__(self, file_path):
        OleFileIO.__init__(self, file_path)
```

```
    def readStream(self, path):
        """ Accept a field string or a list of directories followed by a field string.
        Tries the known data types UNICODE, ASCII and DATA for each field.
        Return the corresponding stream in the OLE object. """
        if isinstance(path, basestring):
            path = [path]  # make sure strings are converted to lists
        field_id = path[-1]  # the last item is the field without data type appended
        for name, enc in encoding.iteritems():  # try data types in order
            path[-1] = "%s%s%s" % (FIELD_DESIGNATOR, field_id, enc)  # replace last item with full field name
            if self.exists(path):
                stream = self.openstream(path).read()
                if name == "UNICODE":
                    print("***** found unicode !")
                    stream = unicode(stream, "utf_16_le")
                return stream
        return None

    def enumerateAttachments(self):
        """ Attachments are stored in folders starting with "__attach".
        List them and create a Message object for each of them and return a list of such objects """
        dirs = []
        for directory in self.listdir():
            if directory[0].startswith(ATTACHMENT_DESIGNATOR) and directory[0] not in dirs:  # can be listed more than once
                dirs.append(directory[0])
        attachments = []
        for directory in dirs:
            attachments.append(Attachment(self, directory))  # create an attachment object from directory
        return attachments


def parse_msg(dirpath, filename):

    FILE_PATH = ("%s/%s") % (dirpath, filename)
    m = Message(FILE_PATH)

    # Check the message type
    message_type = m.readStream(fields['message_type'])
    if EMAIL_TYPE in message_type or MEETING_REQUEST in message_type:
        # This is an e-mail message (and not a task, contact, note or activity)
        # Get the headers, content and attachments
        original_headers = m.readStream(fields['original_headers'])
        content_plain = m.readStream(fields['content_plain'])
        content_html = m.readStream(fields['content_html'])

        # Get the attachments
        attachments = m.enumerateAttachments()

        parsed_original_headers = message_from_string(original_headers)

        # Compose an email
        msg = MIMEMultipart('alternative')  # a multi-part message
        for key in parsed_original_headers.keys():
            if key.lower() == "content-type":
                continue  # the as_string() function will fail if a single part message has multiple parts.

            # replace or add the header key, value pair
            original_header_value = parsed_original_headers[key]
            try:
                msg.replace_header(key, original_header_value)
            except:
                msg.add_header(key, original_header_value)

        # Add plain text if exist
        if content_plain is not None:
            text_part = MIMEText(content_plain, 'plain')
            text_part.set_charset(DEFAULT_CHARSET)
            msg.attach(text_part)

        # Add html version if exist
        if content_html is not None:
            html_part = MIMEText(content_html, 'html')
            html_part.set_charset(DEFAULT_CHARSET)
            msg.attach(html_part)

        # Add attachments if exist
        if len(attachments) > 0:
            for a in attachments:
                attachment_part = MIMEBase('application', 'octet-stream')
                attachment_part.set_payload(a.data)
                encode_base64(attachment_part)
                attachment_part.add_header('Content-Disposition', 'attachment; filename="%s"' % a.name_long)
                attachment_part.add_header('Content-Type', a.mime)
                msg.attach(attachment_part)

        # Write message to file
        target = ("%s/%s.eml") % (dirpath, filename.split(".")[0])
        with open(target, "w") as dest:
            dest.write(msg.as_string())

    else:
        # The object found was not an email message
        print("* Found no email message: %s/%s (type: %s)") % (dirpath, filename, message_type)

    # remove the source file for space conservation
    os.remove(FILE_PATH)
```

E.4 metadata.py

```
"""
This program will read all email per user in plain text format and:
* extract features
* sort the features by date—time
* remove duplicate entries (typically caused by user manually making local copies in multiple directories)
* guess the users own email address based on sender frequency and user name
* extract attachments and store a copy without attachments for further reference
"""

# built in libraries in python 2.7.5
from multiprocessing import Pool, freeze_support, cpu_count
from collections import Counter
from email import message_from_file
from email.header import decode_header
from email.utils import parsedate, parseaddr
from email.parser import HeaderParser
import calendar
import re  # regular expressions
import timeit  # benchmark time
import json  # output format
import md5
import os
import csv

import traceback

# runtime variables
NUM_PROCESSES = cpu_count()  # determine how many cores are available
from server.settings import global_program_settings as global_settings
COMPANY_DOMAIN = global_settings["company_domain"]
INDENT_JSON = global_settings["indent_json"]
EML_MATCH_REGEX = global_settings["eml_match_regex"]
FORWARD_STRING = global_settings["forward_string"]
REPLY_STRING = global_settings["reply_string"]
MAILDIR_PATH = global_settings["maildir_path"]
META_DATA_PATH = global_settings["meta_data_path"]
CONTENT_DATA_PATH = global_settings["content_data_path"]
ATTACHMENT_PATH = global_settings["attachment_path"]
ADDRESS_BOOK_PATH = global_settings["address_book_path"]
CSV_OUTPUT_PATH = global_settings["django_static_root"]

PROCESS_CONTENT_JSON = True
PROCESS_ATTACHMENTS = True

for folder in [META_DATA_PATH, CONTENT_DATA_PATH, ATTACHMENT_PATH]:
    if not os.path.exists(folder):
        os.makedirs(folder)


def convert_unicode(string, encoding, default="ascii"):
    if encoding == "":
        encoding = None
    if encoding is not None:
        return unicode(string, encoding, "replace")
    else:
        return unicode(string, default, "replace")


def html_escape(text):
    """ Replace problematic html characters in message headers
    http://stackoverflow.com/questions/2077283/substitute—special—characters—with—html—entities—in—python """
    html_escape_table = {
        "&": "&amp;",
        '"': "&quot;",
        "'": "&apos;",
        ">": "&gt;",
        "<": "&lt;",
    }
    return "".join(html_escape_table.get(c, c) for c in text)


class EmailMessage:
    def __init__(self, message_path):
        with open(message_path, "rb") as source:  # read only binary
            self.message = message_from_file(source)
            self.attachment_counter = 0
            self.message_size = os.path.getsize(message_path)
            self.structure = self.get_structure()

    def get_structure(self):
        content_types = []
        if self.message.is_multipart():
            for part in self.message.walk():
                content_types.append(part.get_content_type())
        else:  # message is not multipart
            content_types.append(self.message.get_content_type())

        num_multipart, num_plain, num_html, num_other = (0, 0, 0, 0)
        for ct in content_types:
            if 'multipart/' in ct:
                num_multipart += 1
                continue
            if 'text/plain' in ct:
                num_plain += 1
                continue
```

```
            if 'text/html' in ct:
                num_html += 1
                continue
            num_other += 1
        return {
            'multipart': num_multipart,
            'plain': num_plain,
            'html': num_html,
            'other': num_other,
        }

    def header(self, header_id):
        return self.message.get(header_id, "")  # Note: is case insensitive

    def header_decode(self, header):
        header, encoding = decode_header(header)[0]
        if encoding is not None:
            return convert_unicode(header, encoding)
        else:
            return convert_unicode(header, self.get_charset())

    def address_field(self, field):
        """ return a list of participants """
        header = self.header(field)
        header = convert_unicode(header, self.get_charset())
        if header is not "":
            header = header.replace(";", ",")  # comma is the most common separator
            split_address = header.split(",")
            for key, address in enumerate(split_address):
                split_address[key] = parseaddr(address)
            return split_address
        else:
            return [(None, None)]

    def get_charset(self):
        # http://ginstrom.com/scribbles/2007/11/19/parsing-multilingual-email-with-python/
        if self.message.get_content_charset():
            return self.message.get_content_charset()
        if self.message.get_charset():
            return self.message.get_charset()
        return None

    def get_text(self, type="plain"):  # type: "plain" (default) or "html"
        """ return the first instance of plain/text content. Might return None if no such part is found """
        match = False
        content = ""
        if self.message.is_multipart():
            for part in self.message.walk():
                content_type = "text/%s" % type
                char_enc = part.get_content_charset()
                if part.get_content_type().lower() == content_type:
                    match = True
                    #content += convert_unicode(part.get_payload(decode=True), self.get_charset())
                    content += convert_unicode(part.get_payload(decode=True), char_enc)

                    #return part.get_payload(decode=True)
            if match:
                return content
        else:
            return convert_unicode(self.message.get_payload(decode=True), self.get_charset())
            #return convert_unicode(self.message.get_payload(decode=True), self.get_charset())

    def store_attachments(self):
        """ save attachment to a dedicated folder with an ID and name returned in a list """
        found_attachments = []
        if self.message.is_multipart():  # can a non-multipart message have attachments?
            for part in self.message.walk():
                #print part.get_content_maintype()
                if part.get_content_maintype() not in ['multipart', 'text']:
                    filename = part.get_filename()
                    payload = part.get_payload(decode=True)
                    payload_digest = md5.new(payload).hexdigest()
                    target_path = ("%s%s") % (ATTACHMENT_PATH, payload_digest)
                    # do not write a document twice, thus hash name
                    if not os.path.exists(target_path):  # a race condition can happen..
                        with open(target_path, "w") as target:
                            target.write(payload)
                            self.attachment_counter += 1
                    found_attachments.append([filename, payload_digest, part.get_content_type()])
        return found_attachments

    def store_contents(self, username, counter, user_id, source_path, source_file):
        header = {}
        raw_headers = HeaderParser().parsestr(self.message.as_string())
        for name, value in raw_headers.items():
            value = unicode(value, errors='replace')
            header[name] = html_escape(value)

        meta_data_string = u"%s: %s/%s" % (user_id, source_path, source_file)
        header["meta_data_source"] = meta_data_string

        plain_version = self.get_text("plain")
        html_version = self.get_text("html")

        content = {
            "header": header,
            "plain": plain_version,
```

```
            "html": html_version,
        }

        try:
            json.dumps(content)
        except:
            for h in header:
                print type(h)

        target_folder = ("%s%s/") % (CONTENT_DATA_PATH, username)
        if not os.path.exists(target_folder):
            os.makedirs(target_folder)
        target_path = ("%s%s.json") % (target_folder, counter)
        with open(target_path, "w") as dest:
            if INDENT_JSON:
                dest.write(json.dumps(content, indent=1))
            else:
                dest.write(json.dumps(content))
        return counter

    def get_size(self):  # return size of content, html then plaintext
        """ The size of the plain text part found """

        html_data = self.get_text("thml")
        if html_data is not None:
            return len(html_data)

        plain_data = self.get_text("plain")
        if plain_data is not None:
            return len(plain_data)

        return self.message_size  # last resort, return size of file


def worker(username):
    try:
        """  The main loop being run in parallel """
        features = []  # storage of the message features
        subfolders_all = []
        msg_senders_all = Counter()
        verify_counter = 0  # keep track of read emails
        filename_loop_counter = 0  # file name for content json files (used by web server)

        subfolder_path = "%s%s/" % (MAILDIR_PATH, username)
        for (dirpath, dirnames, filenames) in os.walk(subfolder_path):
            for filename in filenames:

                """ Add unique user created folders to a list """
                subfolder = dirpath.split(subfolder_path)[1]
                if subfolder not in subfolders_all:
                    subfolders_all.append(subfolder)

                """ Filter out non email files """
                messageid_match = re.match(EML_MATCH_REGEX, filename)  # integer followed by .eml extension
                if messageid_match:
                    verify_counter += 1  # count the number of valid messages encountered
                    message_path = ("%s/%s") % (dirpath, filename)
                    """ add case meta data """
                    measurements = []  # store information of this message
                    measurements.extend([
                        username,                   # 0
                        subfolder,                  # 1
                        filename                    # 2
                    ])
                else:
                    continue  # break current iteration and continue on next item

                """ Open each message and look for meta data """
                msg = EmailMessage(message_path)

                """ Collect date and time data
                Source: http://stackoverflow.com/questions/1790795/python-parsing-date-with-timezone-from-an-email.
                See https://tools.ietf.org/html/rfc2822#page-20: Not transport time, but when human prepared message for sending """
                msg_date_str = msg.header("Date")
                if msg_date_str == "":
                    print("%s %s %s has no valid date, skipping") % (username, subfolder, filename)
                    continue  # break current iteration and continue on next item

                msg_date_tuple = parsedate(msg_date_str)
                msg_timestamp = calendar.timegm(msg_date_tuple)  # tm_wday, tm_yday and tm_isdst are not used
                # 9-tuple time format [year, month, day, hour, minute, second, weekday, day of year and daylight saving time]
                # example of a 9-tuple time format "(2000, 9, 6, 3, 27, 0, 0, 1, -1)". Note the 3 last to right are not used by parsedate()
                try:
                    msg_timezone = int(re.search('([+-]\d\d)\d\d', str(msg_date_str)).group(1))  # will fail if time zone info does not
                            exist
                except:
                    #print("%s %s %s has no timezone data") % (username, subfolder, filename)
                    msg_timezone = 0

                msg_timestamp_utc = msg_timestamp - (msg_timezone * 3600)  # -1000 means UTC is 10 hours ahead, thus the minus sign bellow
                local_weekday = calendar.weekday(msg_date_tuple[0], msg_date_tuple[1], msg_date_tuple[2])  # 0 indexed, 0 is Monday
                local_hour = msg_date_tuple[3]

                measurements.extend([
                    msg_timestamp,              # 3
                    msg_timestamp_utc,          # 4
                    msg_timezone,               # 5
```

```
            local_weekday,                  # 6
            local_hour                      # 7
        ])

        """ Collect subject data """
        msg_subject = msg.header_decode(msg.header("Subject"))
        msg_subject_length = len(msg_subject)
        msg_subject_forward = False
        msg_subject_reply = False
        if re.search(FORWARD_STRING, msg_subject, re.IGNORECASE):
            msg_subject_forward = True
        if re.search(REPLY_STRING, msg_subject, re.IGNORECASE):
            msg_subject_reply = True

        measurements.extend([
            msg_subject,                    # 8
            msg_subject_length,             # 9
            msg_subject_forward,            # 10
            msg_subject_reply               # 11
        ])

        """ Participants in the conversation """
        msg_sender_name, msg_sender_email = msg.address_field("From")[0]

        msg_to = msg.address_field("To")    # direct copy
        msg_cc = msg.address_field("Cc")    # carbon copy
        msg_bcc = msg.address_field("Bcc")  # blind carbon copy
        #msg_delivered_to = msg.address_field("Delivered-To")

        def add_to_receives(receivers):  # make sure they are unique
            for receiver in receivers:
                name, email = receiver
                if email not in [u'', None]:  # empty receiver pairs are possible
                    #print email
                    if email not in msg_receivers and email is not None:
                        msg_receivers.append(email.lower())

        msg_receivers = []
        add_to_receives(msg_to)
        add_to_receives(msg_cc)
        add_to_receives(msg_bcc)  # in essence, we don't care how the message was received
        #add_to_receives(msg_delivered_to)

        msg_draft = False
        msg_bcc = False
        if msg_sender_email is None or len(msg_receivers) == 0:
            #print("Draft found for %s") % (filename)
            msg_draft = True

        internal_receivers = 0
        external_receivers = 0
        for receiver in msg_receivers:
            #print receiver, COMPANY_DOMAIN
            if re.search(COMPANY_DOMAIN, receiver):
                internal_receivers += 1
            else:
                external_receivers += 1

        measurements.extend([
            internal_receivers,             # 12
            external_receivers,             # 13
            msg_sender_email,               # 14
            msg_receivers,                  # 15
            msg_draft,                      # 16
        ])

        """ size and a list for collecting duplicate folders where the message was found """
        measurements.extend([
            msg.get_size(),                 # 17
            msg.structure['multipart'],     # 18
            msg.structure['plain'],         # 19
            msg.structure['html'],          # 20
            msg.structure['other'],         # 21
            [subfolder]                     # 22  # this is the first not included field when removing duplicates
        ])

        """ ****************************
        Write attachments and content data to files
        **************************** """
        if PROCESS_ATTACHMENTS:
            result = msg.store_attachments()
            measurements.append(result)          # 23
        else:
            measurements.append([])              # 23 alternative

        if PROCESS_CONTENT_JSON:
            result = msg.store_contents(username, filename_loop_counter, measurements[0], measurements[1], measurements[2])
            measurements.append(result)      # 24
        else:
            measurements.append(None)        # 24 alternative
        filename_loop_counter += 1

        """ add all the email features to the data list """
        features.append(measurements)  # end of file

    if len(features) == 0:  # no messages was found: return and skip
        return (0, 0, 0, username, "")
```

```
""" ****************************
Remove duplicate messages
**************************** """
features = sorted(features, key=lambda key: (key[4], key[17]))  # Sort on utc time stamp and the content length

def unique(feature, previous_feature, duplicate_counter):
    """ return True if message is unique and False if same as previous """
    if feature[4] == previous_feature[4]:  # if time stamp (UTC) is equal to the last message: Candidate duplicate item
        match = True
        for index, attribute in enumerate(feature[5:18]):  # skip the first 3 meta data fields (they are unique) and the timestamps
            . Also skip field 18 (folder)
            if attribute != previous_feature[index + 5]:
                match = False  # if any other field is not equal
                break  # no reason to continue
        if match:  # if match is still True (all match)
            duplicate_counter += 1
            return (False, previous_feature, duplicate_counter)
            # Note the only case when we don't update the previous message vector
        else:
            return (True, feature, duplicate_counter)
    else:
        return (True, feature, duplicate_counter)

unique_features = []
previous_feature = [None] * len(features[0])  # default value first iteration
duplicate_counter = 0  # keep track of the number of removed files
discarded_subfolders = []
for f in features:
    is_unique, previous_feature, duplicate_counter = unique(f, previous_feature, duplicate_counter)  # only keep the ones that
        return True (unique)
    if is_unique:
        unique_features.append(f)
        msg_senders_all[f[14]] += 1  # count the sender email in order to determine direction of message
        if len(discarded_subfolders) > 0:
            unique_features[-2][22].extend(discarded_subfolders)  # the previous one is updated
        discarded_subfolders = []  # reset
    else:
        discarded_subfolders.append(f[22][0])  # append the subfolder of the discarded duplicate message
        # remove the content json file
        if PROCESS_CONTENT_JSON:
            meta_data_path = "%s%s/%s.json" % (CONTENT_DATA_PATH, username, f[24])  # 24 is the returned json file id (content_id)
            os.remove(meta_data_path)

""" ****************************
Determine users own email address:
**************************** """
current_user_email = None
username_stripped_domain = username.split("@")[0]
try:
    user_domain = ("@%s") % username.split("@")[1]
except:
    user_domain = COMPANY_DOMAIN
lastname = username_stripped_domain.split("-")[0]
match_string = "%s%s$" % (lastname, user_domain)
for address, count in msg_senders_all.most_common():
    if re.search(match_string, address):
        current_user_email = address
        break
if current_user_email is None:
    print ("*** GUESSING THE FOLLOWING EMAIL ADDRESS ***")
    current_user_email = msg_senders_all.most_common(1)[0][0]

if COMPANY_DOMAIN == "@enron.com":  # one exception where both methods fail for the Enron dataset
    if username == "merriss-s":
        print ("*** CORRECTING GUESS FOR merriss-s ***")
        current_user_email = "steven.merris@enron.com"

""" ****************************
Assemble the JSON data structure:
**************************** """
data = {
    'username': username,
    'e-mail': current_user_email,
    'duplicate_messages': duplicate_counter,
    'original_messages': verify_counter,
    'data': unique_features,
    'original_account': True,
    'attributes': [
        ('meta_username', 'STRING'),           # 0
        ('meta_folder', subfolders_all),       # 1
        ('meta_messageid', 'STRING'),          # 2
        ('timestamp_local', 'INTEGER'),        # 3
        ('timestamp_utc', 'INTEGER'),          # 4
        ('timezone', 'INTEGER'),               # 5
        ('local_weekday', 'INTEGER'),          # 6
        ('local_hour', 'INTEGER'),             # 7
        ('email_subject', 'STRING'),           # 8
        ('email_subject_length', 'INTEGER'),   # 9
        ('email_subject_forward', 'BOOLEAN'),  # 10
        ('email_subject_reply', 'BOOLEAN'),    # 11
        ('internal_receivers', 'INTEGER'),     # 12
        ('external_receivers', 'INTEGER'),     # 13
        ('email_sender', 'STRING'),            # 14
        ('email_receivers', 'LIST'),           # 15
        ('draft', 'BOOLEAN'),                  # 16
        ('email_payload_len', 'INTEGER'),      # 17
```

```
                ('parts_multipart', 'INTEGER'),         # 18
                ('parts_plain', 'INTEGER'),             # 19
                ('parts_html', 'INTEGER'),              # 20
                ('parts_other', 'INTEGER'),             # 21
                ('email_folders', 'LIST'),              # 22
                ('attachments', 'LIST'),                # 23
                ('content_id', 'INTEGER')               # 24
            ],
        }

        """ ****************************
        Write JSON to file:
        **************************** """
        target = "%s%s.json" % (META_DATA_PATH, username)
        with open(target, "w") as dest:  # (over)write mode
            if INDENT_JSON:
                dest.write(json.dumps(data, indent=1))
            else:
                dest.write(json.dumps(data))

        """ ****************************
        Summarize and return statistics:
        **************************** """
        num_imported = len(unique_features)
        print "\t* %s\n\t\t%s of %s messages extracted.\n\t\t%s duplicates removed.\n\t\tAddress: %s" % (
            username,
            num_imported,
            verify_counter,
            duplicate_counter,
            current_user_email
        )

        return {
            "verify_counter": verify_counter,
            "num_imported": num_imported,
            "duplicate_counter": duplicate_counter,
            "username": username,
            "current_user_email": current_user_email,
            "counter": msg_senders_all,
        }
    except:
        traceback.print_exc()


### speed up code
def main():
    usernames = []
    for (dirpath, dirnames, filenames) in os.walk(MAILDIR_PATH):
        usernames.extend(dirnames)
        break  # do not continue walking
    print("* Found %s usernames") % (len(usernames))
    usernames = sorted(usernames)

    total_number_messages = 0
    total_number_messages_imported = 0
    total_number_duplicates = 0
    address_book = {}  # place to store the look-up table between addresses and usernames
    combined_msg_senders = []

    print("* Parsing users email in parallel..")
    pool = Pool(processes=NUM_PROCESSES)
    for result in pool.imap_unordered(worker, usernames, chunksize=2):
        total_number_messages += result["verify_counter"]
        total_number_messages_imported += result["num_imported"]
        total_number_duplicates += result["duplicate_counter"]
        address_book[result["current_user_email"]] = result["username"]
        combined_msg_senders.append({
                                    "counter": result["counter"],
                                    "address": result["current_user_email"],
                                    "username": result["username"]
                                    })
    pool.close()
    pool.join()

    print("* Writing the address book..")
    with open(ADDRESS_BOOK_PATH, "w") as dest:
        dest.write(json.dumps(address_book))

    csv_save_path = "%ssender_statistics.csv" % (CSV_OUTPUT_PATH)
    with open(csv_save_path, "w") as destination:
        writer = csv.writer(destination, delimiter="|", quotechar='"', quoting=csv.QUOTE_NONNUMERIC)
        writer.writerow(("identifier", "count", "sender_address"))
        for counter in combined_msg_senders:
            userstr = "%s (%s)" % (counter["username"], counter["address"])
            for count, addr in counter["counter"].items():
                writer.writerow((userstr, addr, count))

    print("* All work has completed!\n* %s of %s messages extracted.\n* %s duplicates removed.") % (total_number_messages_imported,
          total_number_messages, total_number_duplicates)


if __name__ == '__main__':
    # benchmark runtime
    print("** Reading from %s") % (MAILDIR_PATH)
    timer_start = timeit.default_timer()
    freeze_support()
    main()
```

```
    timer_stop = timeit.default_timer()
    print("** Completed in %ss") % round(timer_stop - timer_start, 2)
```

E.5 direction.py

```
"""
This script will run through all feature files from metadata.py
It will decide on which direction the message was sent, to or from the current user.
It will then based on the direction determine if another the message was sent to or received from another user
It is assumed the messages have been sorted by date.
"""

# built in libraries in python 2.7.5
from multiprocessing import Pool, freeze_support, cpu_count
import timeit  # benchmark time
import json
import os

# runtime variables
NUM_PROCESSES = cpu_count()
from server.settings import global_program_settings as global_settings
INDENT_JSON = global_settings["indent_json"]
META_DATA_PATH = global_settings["meta_data_path"]
ADDRESS_BOOK_PATH = global_settings["address_book_path"]
CANDIDATE_MESSAGES_PATH = global_settings["candidate_messages_path"]


def worker(feature_file_path):
    candidate_lookup_set = []
    with open(feature_file_path, "rb") as source:
        input_data = json.load(source)

    with open(ADDRESS_BOOK_PATH, "rb") as source:
        address_book = json.load(source)

    attributes = input_data['attributes']

    def id_from_name(name):
        for index, attribute in enumerate(attributes):
            if attribute[0] == name:
                return index
        return False

    # input file mappings
    EMAIL_FROM_ID = id_from_name("email_sender")
    EMAIL_TO_ID = id_from_name("email_receivers")

    user_email = input_data['e-mail']

    """ Loop all messages for current user and gather data """
    for mid, message in enumerate(input_data['data']):

        """ Determine direction based on current user """
        email_sent = False  # True: sent by user, False: email was received
        if message[EMAIL_FROM_ID] == user_email:  # email_from == user_email
            email_sent = True

        # address_book determine suspect in list
        other_party_suspect = False
        if email_sent:
            compare_field = message[EMAIL_TO_ID]
        else:
            # message[EMAIL_FROM_ID] is a string. Converted to list.
            compare_field = [message[EMAIL_FROM_ID]] + message[EMAIL_TO_ID]

        for known_user in address_book:
            if known_user in compare_field:
                other_party_suspect = True
                candidate_lookup_set.append(message)
                break  # just need one example to conclude

        email_missing = False
        if user_email not in message[EMAIL_TO_ID] + [message[EMAIL_FROM_ID]]:
            email_missing = True

        message.append(email_sent)
        message.append(other_party_suspect)
        message.append(email_missing)

    """ Add newly appended attribute descriptions """
    attributes.append(('email_sent', 'BOOLEAN'))            # 25
    attributes.append(('other_party_suspect', 'BOOLEAN'))   # 26
    attributes.append(('email_missing', 'BOOLEAN'))         # 27

    """ Re-save the feature file """
    with open(feature_file_path, "w") as dest:  # append mode
        if INDENT_JSON:
            dest.write(json.dumps(input_data, indent=1))
        else:
            dest.write(json.dumps(input_data))

    return candidate_lookup_set
```

```
### speed up code
def main():
    print("* Determining direction")
    candidate_lookup_set = []
    work_queue = []
    for (dirpath, dirnames, filenames) in os.walk(META_DATA_PATH):
        for filename in filenames:
            if filename.split(".")[-1] == "json":
                feature_file_path = ("%s%s") % (dirpath, filename)
                work_queue.append(feature_file_path)

    pool = Pool(processes=NUM_PROCESSES)
    for result in pool.imap_unordered(worker, work_queue, chunksize=2):
        candidate_lookup_set.extend(result)  # this won't scale to very large datasets
    pool.close()
    pool.join()

    with open(work_queue[0], "rb") as source:
        input_data = json.load(source)

    def id_from_name(name):
        for index, attribute in enumerate(input_data['attributes']):
            if attribute[0] == name:
                return index
        return False

    TIMESTAMP_UTC_ID = id_from_name("timestamp_utc")
    INTERNAL_RECEIVERS_ID = id_from_name("internal_receivers")
    EMAIL_PAYLOAD_LEN_ID = id_from_name("email_payload_len")
    META_USERNAME_ID = id_from_name("meta_username")

    """ ****************************
    Remove duplicate messages
    **************************** """
    candidate_lookup_set = sorted(candidate_lookup_set, key=lambda key: (key[TIMESTAMP_UTC_ID], key[EMAIL_PAYLOAD_LEN_ID]))  # Sort on utc
        time stamp and the content length

    def unique(feature, previous_feature, duplicate_counter):
        """ return True if message is unique and False if same as previous """
        if feature[4] == previous_feature[4]:  # if time stamp (UTC) is equal to the last message: Candidate duplicate item
            match = True
            for index, attribute in enumerate(feature[5:18]):  # skip the first 3 meta data fields (they are unique) and the timestamps.
                Also skip field 18 (folder)
                if attribute != previous_feature[index + 5]:
                    match = False  # if any other field is not equal
                    break  # no reason to continue
            if match:  # if match is still True (all match)
                duplicate_counter += 1
                return (False, previous_feature, duplicate_counter)
                # Note the only case when we don't update the previous message vector
            else:
                return (True, feature, duplicate_counter)
        else:
            return (True, feature, duplicate_counter)

    unique_messages = []
    previous_feature = [None] * len(candidate_lookup_set[0])  # default value first iteration
    duplicate_counter = 0  # keep track of the number of removed files
    for f in candidate_lookup_set:
        is_unique, previous_feature, duplicate_counter = unique(f, previous_feature, duplicate_counter)  # only keep the ones that return
            True (unique)
        if is_unique:
            unique_messages.append(f)

    """ ****************************
    Remove duplicate messages
    **************************** """
    candidate_lookup_set = sorted(unique_messages, key=lambda key: (key[TIMESTAMP_UTC_ID], key[INTERNAL_RECEIVERS_ID], key[META_USERNAME_ID
        ]))

    print("* Writing candidate deleted messages")
    with open(CANDIDATE_MESSAGES_PATH, "w") as dest:
        dest.write(json.dumps(candidate_lookup_set))

if __name__ == '__main__':
    timer_start = timeit.default_timer()
    freeze_support()
    main()
    timer_stop = timeit.default_timer()
    print("** Completed in %ss") % round(timer_stop - timer_start, 2)
```

E.6 undelete.py

```
"""
This script will load potential candidates of deleted messages in others email accounts and add back when missing
It is run after direction.py is complete
"""

# built in libraries in python 2.7.5
from multiprocessing import Pool, freeze_support, cpu_count
import timeit  # benchmark time
import json
import os

# runtime variables
NUM_PROCESSES = cpu_count()
from server.settings import global_program_settings as global_settings
INDENT_JSON = global_settings["indent_json"]
META_DATA_PATH = global_settings["meta_data_path"]
CANDIDATE_MESSAGES_PATH = global_settings["candidate_messages_path"]
OVERVIEW_STATISTICS_PATH = global_settings["overview_statistics_path"]


def undelete(feature_file_path):
    # read in a feature file
    with open(feature_file_path, "rb") as source:
        input_data = json.load(source)

    with open(CANDIDATE_MESSAGES_PATH, "rb") as source:
        candidate_messages = json.load(source)

    def id_from_name(name):
        for index, attribute in enumerate(input_data['attributes']):
            if attribute[0] == name:
                return index
        return False

    #username = input_data['username']
    user_email = input_data['e-mail']
    from_field_id = id_from_name('email_sender')
    to_field_id = id_from_name('email_receivers')
    timestamp_local_id = id_from_name('timestamp_local')
    email_subject_id = id_from_name("email_subject")
    email_payload_len_id = id_from_name("email_payload_len")
    was_sent_id = id_from_name("email_sent")
    email_folders_id = id_from_name("email_folders")
    missmatch_id = id_from_name("email_missing")

    own_messages = input_data['data']

    def same_date(timestamp):
        own_candidates = []
        for message in own_messages:
            if message[timestamp_local_id] == timestamp:
                own_candidates.append(message)
        return own_candidates

    num_messages_added = [0]  # an integer results in "defined out of scope error" thus an array.
    new_messages = []

    # set deleted flag on all of them, revert sent flag and add message
    def check_email(target, reverse=True):
        if user_email in target:
            own_messages_same_date = same_date(candidate[timestamp_local_id])
            match = True
            for owm in own_messages_same_date:
                if (owm[email_subject_id] == candidate[email_subject_id]
                        and owm[email_payload_len_id] == candidate[email_payload_len_id]):
                    match = False
                    break
            if match:
                if reverse:
                    candidate[was_sent_id] = not candidate[was_sent_id]  # reverse direction when adding
                candidate[email_folders_id] = []  # empty the folder location
                candidate[missmatch_id] = False  # This flag is not valid anymore and it cannot be a mismatch by definition
                if len(candidate) == attribute_length:
                    candidate.append(True)  # it has been added, and only if this message has not been appended before
                new_messages.append(candidate)
                num_messages_added[0] += 1

    """ Loop """
    try:
        attribute_length = len(candidate_messages[0])
    except:
        attribute_length = None
    for candidate in candidate_messages:
        # add all matches To/From
        if candidate[was_sent_id]:  # was sent by that other user
            check_email(candidate[to_field_id])
        else:  # message was received by that other user
            check_email([candidate[from_field_id]])  # must be a list
            # the current user can be in the To field even if the message was received
            check_email(candidate[to_field_id], False)

    for message in own_messages:
        message.append(False)  # add a flag: "added back" for all currently present

    own_messages.extend(new_messages)
```

```
    input_data['data'] = sorted(own_messages, key=lambda key: (key[timestamp_local_id], key[email_payload_len_id]))  # Sort on time stamp
           and the BODY length

    input_data['attributes'].append(('added_back', 'BOOLEAN'))  # 28
    input_data['put_back'] = num_messages_added[0]

    """ Re-save the feature file """
    with open(feature_file_path, "w") as dest:  # append mode
        if INDENT_JSON:
            dest.write(json.dumps(input_data, indent=1))
        else:
            dest.write(json.dumps(input_data))
    #print("Added %s messages to %s") % (num_messages_added[0], username)

    return num_messages_added[0]
    ### WORKER DONE


### speed up code
def main():
    work_queue = []
    for (dirpath, dirnames, filenames) in os.walk(META_DATA_PATH):
        for filename in filenames:
            if filename.split(".")[-1] == "json":
                feature_file_path = ("%s%s") % (dirpath, filename)
                work_queue.append(feature_file_path)

    print("* Adding back deleted messages")
    total_num_messages_added = 0
    pool = Pool(processes=NUM_PROCESSES)
    for result in pool.imap_unordered(undelete, work_queue):
        total_num_messages_added += result
    pool.close()
    pool.join()
    print("* Added back %s deleted messages in total") % total_num_messages_added


if __name__ == '__main__':
    # benchmark runtime
    timer_start = timeit.default_timer()
    freeze_support()
    main()
    timer_stop = timeit.default_timer()
    print("** Completed in %ss") % round(timer_stop - timer_start, 2)
```

E.7 timestatistics.py

```
"""This script will add messages per day, per week. It will also add unique peers per day.
It is run after undelete.py
"""

from multiprocessing import Pool, freeze_support, cpu_count
import timeit  # benchmark time
import json
import os

# runtime variables
NUM_PROCESSES = cpu_count()
from server.settings import global_program_settings as global_settings
INDENT_JSON = global_settings["indent_json"]
META_DATA_PATH = global_settings["meta_data_path"]
CANDIDATE_MESSAGES_PATH = global_settings["candidate_messages_path"]


def timestats(feature_file_path):
    from collections import Counter
    import datetime

    # read in a feature file
    with open(feature_file_path, "rb") as source:
        input_data = json.load(source)

    def id_from_name(name):
        for index, attribute in enumerate(input_data['attributes']):
            if attribute[0] == name:
                return index
        return False

    # input file mappings
    EMAIL_FROM_ID = id_from_name("email_sender")
    EMAIL_TO_ID = id_from_name("email_receivers")
    TIMESTAMP_LOCAL_ID = id_from_name("timestamp_local")
    DIRECTION_ID = id_from_name("email_sent")

    attributes = input_data['attributes']
    try:
        last_message_id = len(input_data['data']) - 1  # -1: make it zero indexed
    except:
        print("Timestatistics: No messages found, quiting")
        return

    cnt_day = Counter()
    cnt_week = Counter()
    ppl_day_storage = {}
    cnt_ppl_day_sent = []
    cnt_ppl_day_rece = []
```

```
    previous_message_day = None

    """ Loop all messages for current user and gather data """
    for message_id, message in enumerate(input_data['data']):

        email_sent = message[DIRECTION_ID]

        """ Determine number of messages per day and week in both directions.
        The combination year, week and day + direction will be unique for each day and week """
        timestamp = int(message[TIMESTAMP_LOCAL_ID])
        date = datetime.datetime.fromtimestamp(timestamp)
        year, week, day = date.isocalendar()

        unique_day = ("%s-%s-%s-%s") % (year, week, day, email_sent)
        unique_week = ("%s-%s-%s") % (year, week, email_sent)
        cnt_day[unique_day] += 1
        cnt_week[unique_week] += 1

        """ Determine number of unique people talked to """
        def count_and_save(previous_message_day, cnt_ppl_day_sent, cnt_ppl_day_rece):
            year, week, day = previous_message_day
            unique_last_day_sent = ("%s-%s-%s-%s") % (year, week, day, True)
            unique_last_day_rece = ("%s-%s-%s-%s") % (year, week, day, False)
            ppl_day_storage[unique_last_day_sent] = len(set(cnt_ppl_day_sent))  # length unique
            ppl_day_storage[unique_last_day_rece] = len(set(cnt_ppl_day_rece))  # length unique

        current_day = (year, week, day)

        if previous_message_day and (current_day != previous_message_day):
            count_and_save(previous_message_day, cnt_ppl_day_sent, cnt_ppl_day_rece)
            cnt_ppl_day_sent = []  # reset counter
            cnt_ppl_day_rece = []  # reset counter

        if email_sent:  # add email addresses
            for addr in message[EMAIL_TO_ID]:  # can be many recipients
                cnt_ppl_day_sent.append(addr)
        else:  # received...
            cnt_ppl_day_rece.append(message[EMAIL_FROM_ID])  # can only be one sender

        if message_id == last_message_id:  # the store function is only called on week change, so this is a special case
            count_and_save(current_day, cnt_ppl_day_sent, cnt_ppl_day_rece)

        previous_message_day = current_day  # update the year and week for next round (last thing to happen)

    """ Loop all messages in order to fill inn statistics from last round"""
    for message in input_data['data']:
        timestamp = int(message[TIMESTAMP_LOCAL_ID])
        date = datetime.datetime.fromtimestamp(timestamp)
        year, week, day = date.isocalendar()

        unique_day = ("%s-%s-%s-%s") % (year, week, day, message[DIRECTION_ID])
        unique_week = ("%s-%s-%s") % (year, week, message[DIRECTION_ID])

        message.append(cnt_day[unique_day])
        message.append(cnt_week[unique_week])
        message.append(ppl_day_storage[unique_day])

    """ Add newly appended attribute descriptions """
    attributes.append(('messages_day', 'INTEGER'))          # 29
    attributes.append(('messages_week', 'INTEGER'))         # 30
    attributes.append(('unique_people_day', 'INTEGER'))     # 31

    """ Re-save the feature file """
    with open(feature_file_path, "w") as dest:  # append mode
        if INDENT_JSON:
            dest.write(json.dumps(input_data, indent=1))
        else:
            dest.write(json.dumps(input_data))

    return True


### speed up code
def main():
    print("* Adding time statistics")
    work_queue = []
    for (dirpath, dirnames, filenames) in os.walk(META_DATA_PATH):
        for filename in filenames:
            if filename.split(".")[-1] == "json":
                feature_file_path = ("%s%s") % (dirpath, filename)
                work_queue.append(feature_file_path)

    pool = Pool(processes=NUM_PROCESSES)
    for result in pool.imap_unordered(timestats, work_queue):
        pass
    pool.close()
    pool.join()

if __name__ == '__main__':
    # benchmark runtime
    timer_start = timeit.default_timer()
    freeze_support()
    main()
    timer_stop = timeit.default_timer()
    print("** Completed in %ss") % round(timer_stop - timer_start, 2)
```

E.8 recover.py

```
#This script will try to rebuild a non—existing users email account based on messages left in the known accounts """
import timeit  # benchmark time
import json
import os

# runtime variables
from server.settings import global_program_settings as global_settings
INDENT_JSON = global_settings["indent_json"]
META_DATA_PATH = global_settings["meta_data_path"]
ADDRESS_BOOK_PATH = global_settings["address_book_path"]
OVERVIEW_STATISTICS_PATH = global_settings["overview_statistics_path"]


def simplity_address(address):
    try:
        email_user = address.split("@")[0].replace(".", "").split("+")[0]
        email_domain = address.split("@")[1]
        return email_user + "@" + email_domain
    except:
        return address


def worker(restore_username, restore_email):
    # read all known accounts
    restore_email = simplity_address(restore_email)
    restored_messages = []
    print("* scanning messages for account restoration")
    for (dirpath, dirnames, filenames) in os.walk(META_DATA_PATH):
        for filename in filenames:
            if filename.split(".")[—1] == "json":
                feature_file_path = ("%s%s") % (dirpath, filename)

                with open(feature_file_path, "rb") as source:
                    input_data = json.load(source)

                if input_data['original_account'] is False:  # another recovered account
                    continue  # skip to next file

                def id_from_name(name):
                    for index, attribute in enumerate(input_data['attributes']):
                        if attribute[0] == name:
                            return index
                    return False

                from_field_id = id_from_name('email_sender')
                to_field_id = id_from_name('email_receivers')
                was_sent_id = id_from_name("email_sent")
                email_folders_id = id_from_name('email_folders')
                other_party_suspect_id = id_from_name('other_party_suspect')
                missmatch_id = id_from_name("email_missing")
                added_back_id = id_from_name("added_back")

                # set deleted flag on all of them, revert sent flag and add message
                def check_email(target, reverse=True):
                    if restore_email in [simplity_address(address) for address in target]:
                        if reverse:
                            message[was_sent_id] = not message[was_sent_id]  # invert
                        message[email_folders_id] = []  # reset

                        ### DRAFT MUST BE REDETERMINED

                        message[missmatch_id] = False  # This flag is not valid anymore
                        message[other_party_suspect_id] = True  # must be another suspect
                        message[added_back_id] = True  # was added back
                        restored_messages.append(message[0:added_back_id + 1])  # don't include any time statistic

                # add all matches To/From
                for message in input_data["data"]:
                    if message[added_back_id] is True:
                        continue
                    if message[was_sent_id]:  # was sent by that other user
                        check_email(message[to_field_id])
                    else:  # message was received by that other user
                        check_email([message[from_field_id]])  # must be a list
                        # the current user can be in the To field even if the message was received
                        check_email(message[to_field_id], False)

    if len(restored_messages) == 0:
        print("* Did not find any messages for the address %s") % (restore_email)
        return False

    timestamp_utc_id = 4
    email_payload_len = 17
    email_from_id = 14

    """ ****************************
    Remove duplicate messages
    **************************** """
    # Sort on utc time stamp and the content length then the sender
    restored_messages = sorted(restored_messages, key=lambda key: (key[timestamp_utc_id], key[email_payload_len], key[email_from_id]))

    def unique(feature, previous_feature, duplicate_counter):
        """ return True if message is unique and False if same as previous """
        if feature[4] == previous_feature[4]:  # if time stamp (UTC) is equal to the last message: Candidate duplicate item
            match = True
```

```
            for index, attribute in enumerate(feature[5:18]):  # skip the first 3 meta data fields (they are unique) and the timestamps.
                  Also skip field 18 (folder)
                if attribute != previous_feature[index + 5]:
                    match = False  # if any other field is not equal
                    break  # no reason to continue
            if match:  # if match is still True (all match)
                duplicate_counter += 1
                return (False, previous_feature, duplicate_counter)
                # Note the only case when we don't update the previous message vector
            else:
                return (True, feature, duplicate_counter)
        else:
            return (True, feature, duplicate_counter)

    unique_messages = []
    previous_feature = [None] * len(restored_messages[0])  # default value first iteration
    duplicate_counter = 0  # keep track of the number of removed files
    for f in restored_messages:
        is_unique, previous_feature, duplicate_counter = unique(f, previous_feature, duplicate_counter)  # only keep the ones that return
             True (unique)
        if is_unique:
            unique_messages.append(f)

    """ ****************************
    Remove duplicate messages
    **************************** """

    restored_messages = sorted(unique_messages, key=lambda key: (key[timestamp_utc_id], key[email_payload_len]))  # Sort on utc time stamp
          and the content length

    """ ****************************
    Assemble the JSON data structure:
    **************************** """
    data = {
        'username': restore_username,
        'e-mail': restore_email,
        'duplicate_messages': 0,
        'data': restored_messages,
        'original_messages': 0,
        'put_back': len(restored_messages),
        'original_account': False,
        'attributes': [
            ('meta_username', 'STRING'),            # 0
            ('meta_folder', []),                    # 1
            ('meta_messageid', 'STRING'),           # 2
            ('timestamp_local', 'INTEGER'),         # 3
            ('timestamp_utc', 'INTEGER'),           # 4
            ('timezone', 'INTEGER'),                # 5
            ('local_weekday', 'INTEGER'),           # 6
            ('local_hour', 'INTEGER'),              # 7
            ('email_subject', 'STRING'),            # 8
            ('email_subject_length', 'INTEGER'),    # 9
            ('email_subject_forward', 'BOOLEAN'),   # 10
            ('email_subject_reply', 'BOOLEAN'),     # 11
            ('internal_receivers', 'INTEGER'),      # 12
            ('external_receivers', 'INTEGER'),      # 13
            ('email_sender', 'STRING'),             # 14
            ('email_receivers', 'LIST'),            # 15
            ('draft', 'BOOLEAN'),                   # 16
            ('email_payload_len', 'INTEGER'),       # 17
            ('parts_multipart', 'INTEGER'),         # 18
            ('parts_plain', 'INTEGER'),             # 19
            ('parts_html', 'INTEGER'),              # 20
            ('parts_other', 'INTEGER'),             # 21
            ('email_folders', 'LIST'),              # 22
            ('attachments', 'LIST'),                # 23
            ('content_id', 'INTEGER'),              # 24
            ('email_sent', 'BOOLEAN'),              # 25
            ('other_party_suspect', 'BOOLEAN'),     # 26
            ('email_missing', 'BOOLEAN'),           # 27
            ('added_back', 'BOOLEAN'),              # 28
        ],
    }

    # save to json with the other users
    target = "%s%s.%s" % (META_DATA_PATH, restore_username, "json")
    with open(target, "w") as dest:  # append mode
        if INDENT_JSON:
            dest.write(json.dumps(data, indent=1))
        else:
            dest.write(json.dumps(data))
    print("Added %s messages to %s") % (len(restored_messages), restore_username)
    return target


if __name__ == '__main__':
    restore_email = raw_input("Please enter an email address for restoration:")
    print("* Looking for messages involving %s") % restore_email

    # benchmark runtime
    timer_start = timeit.default_timer()
    with open(ADDRESS_BOOK_PATH, "rb") as source:
        address_book = json.load(source)
    if restore_email not in address_book:
        restore_username = restore_email.split("@")[0]
        meta_json_path = worker(restore_username, restore_email)
        if meta_json_path is not False:
```

```
            # add to address book
            print("* Updating the address book")
            address_book[restore_email] = restore_username
            with open(ADDRESS_BOOK_PATH, "w") as dest:
                dest.write(json.dumps(address_book))

            # run timestatistics on this
            print("* Calculating time statistics")
            from timestatistics import timestats
            timestats(meta_json_path)

    else:
        print("* The email address already exist in the address book, quiting.")
    timer_stop = timeit.default_timer()
    print("** Completed in %ss") % round(timer_stop - timer_start, 2)
```

E.9 tableau.py

```
""" This program will convert meta data from JSON files to |SV after converting the date to string and merge everything to a combined |SV
    file.
"""

import timeit  # benchmark time
import json
import csv
import os
import datetime


from server.settings import global_program_settings as global_settings
META_DATA_PATH = global_settings["meta_data_path"]
OUTPUT_PATH = global_settings["django_static_root"]

output_dir = "%stableau_csv/" % (OUTPUT_PATH)
if not os.path.exists(output_dir):
    os.makedirs(output_dir)


def csv_writer(destination):
    return csv.writer(destination, delimiter="|", quotechar='"', quoting=csv.QUOTE_NONNUMERIC)


def toDateString(timestamp):
    return datetime.datetime.fromtimestamp(timestamp).strftime('%Y-%m-%d %H:%M:%S')


def worker(attributes):
    csv_save_path = "%s%s.csv" % (output_dir, attributes["username"])

    #lookup table
    def id_from_name(name):
        for index, attribute in enumerate(attributes['attributes']):
            if attribute[0] == name:
                return index
        return None

    timestamp_local_id = id_from_name("timestamp_local")
    timestamp_utc_id = id_from_name("timestamp_utc")
    timestamp_utc_id = id_from_name("timestamp_utc")
    email_subject_id = id_from_name("email_subject")
    email_receivers_id = id_from_name("email_receivers")
    email_folders_id = id_from_name("email_folders")
    attachments_id = id_from_name("attachments")
    #add required identifiers here

    #replace timestamps
    for row in attributes["data"]:
        row[timestamp_local_id] = toDateString(row[timestamp_local_id])
        row[timestamp_utc_id] = toDateString(row[timestamp_utc_id])
        row[email_subject_id] = ""  # not needed at all

        #add other modifications here...
        #row[email_receivers_id] = ""
        #row[email_folders_id] = ""
        #row[attachments_id] = ""

        row.append(attributes["username"])  # needed for tracking what file the data was fetched from

    with open(csv_save_path, "w") as destination:
        writer = csv_writer(destination)
        writer.writerows(attributes["data"])


def main():
    work_queue = []
    for (dirpath, dirnames, filenames) in os.walk(META_DATA_PATH):
        for filename in filenames:
            if filename.split(".")[-1] == "json":  # only json files
                feature_file_path = ("%s%s") % (dirpath, filename)
                work_queue.append(feature_file_path)

    for path in work_queue:
        with open(path, "rb") as source:
            attributes = json.load(source)
```

```
            worker(attributes)

    fieldnames = [name for name, type in attributes["attributes"]]
    fieldnames.append("cvs_owner")

    #create a combined file
    csv_save_path = "%sall_combined.csv" % (output_dir)
    with open(csv_save_path, "w") as destination:
        writer = csv_writer(destination)
        writer.writerow(fieldnames)

    csv_files = []
    for (dirpath, dirnames, filenames) in os.walk(output_dir):
        for filename in filenames:
            if filename.split(".")[-1] == "csv":  # only csv files
                feature_file_path = ("%s%s") % (dirpath, filename)
                if feature_file_path != csv_save_path:
                    csv_files.append(feature_file_path)

    for filepath in csv_files:
        with open(filepath, "r") as source:  # read
            with open(csv_save_path, "a") as destination:  # append
                destination.write(source.read())


if __name__ == '__main__':
    timer_start = timeit.default_timer()
    main()
    timer_stop = timeit.default_timer()
    print("** Completed in %ss") % round(timer_stop - timer_start, 2)
```

E.10 server/settings.py

```
root_base = '/path/to/project/folder/'  # NEEDS trailing slash
parsed_content_folder = "processed/"

global_program_settings = {
    "company_domain": '@enron.com',  # for what is assumed "internal email"
    "indent_json": False,  # make json output readable by mixing in newlines
    "eml_match_regex": "(\d+)",  # Used for Enron since it does not have extensions
    #"eml_match_regex": "(\d+\.eml)",  # Used for PST files since they do have .eml extension
    "forward_string": "fwd?:",  # regex ("fw:" or "fwd:")
    "reply_string": "re:",  # regex  ("re:")
    "pst_path": "%spst/" % root_base,
    "maildir_path": "%smaildir/" % root_base,
    "django_static_root": "%s%s" % (root_base, parsed_content_folder),
    "meta_data_path": "%s%smetadata/" % (root_base, parsed_content_folder),  # trailing slash
    "attachment_path": "%s%sattachments/" % (root_base, parsed_content_folder),  # trailing slash
    "content_data_path": "%s%scontent/" % (root_base, parsed_content_folder),  # trailing slash
    "address_book_path": "%s%saddress_book.json" % (root_base, parsed_content_folder),
    "address_book_static_path": "/static/address_book.json",
    "candidate_messages_path": "%s%scandidate_messages.json" % (root_base, parsed_content_folder),
    "overview_statistics_path": "%s%sall_users.json" % (root_base, parsed_content_folder),
}
# remaining is skipped.
```

E.11 server/urls.py

```
from django.conf.urls import patterns, url

urlpatterns = patterns(
    '',
    url(r'^$', 'server.views.overview', name='overview'),
    url(r'^usertimeline/(?P<user_id>[^/]+)/$', 'server.views.usertimeline', name='usertimeline'),
    url(r'^usercom_sent/(?P<user_id>[^/]+)/$', 'server.views.usercom', {'direction': True}, name='usercom_sent'),
    url(r'^usercom_received/(?P<user_id>[^/]+)/$', 'server.views.usercom', {'direction': False}, name='usercom_received'),
    url(r'^usertod/(?P<user_id>[^/]+)/$', 'server.views.usertod', name='usertod'),
    url(r'^messages/(?P<user_id>[^/]+)/$', 'server.views.messages', name='messages'),
    url(r'^messages/(?P<user_id>[^/]+)/(?P<timestamp>\d+)/$', 'server.views.messages', name='messages_date'),
    url(r'^mark_message/(?P<user_id>[^/]+)/(?P<file_id>\d+)/$', 'server.views.mark_message', name='mark_message'),
    url(r'^case_timeline/$', 'server.views.case_timeline', name='case_timeline'),
    url(r'^add_casetimeline/$', 'server.views.add_casetimeline', name='add_casetimeline'),
)
```

E.12 server/views.py

```
from django.shortcuts import render_to_response
from django.template import RequestContext
from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt
from settings import global_program_settings
import json
from models import ReadStatus, CaseTimeline


def overview(request):
    from datetime import datetime
    from overview import main
    statistics = main()
    statistics = sorted(statistics, key=lambda key: key["address"])  # reverse=True can be used

    set_timestamp_low = None
    set_timestamp_high = None
    for account in statistics:
        if account["timestamp_low"] < set_timestamp_low or set_timestamp_low is None:
            set_timestamp_low = account["timestamp_low"]
        if account["timestamp_high"] > set_timestamp_high or set_timestamp_high is None:
            set_timestamp_high = account["timestamp_high"]

    def ts_to_yearmonth(ts):
        date = datetime.fromtimestamp(ts)
        year = date.year
        month = date.month
        if month < 10:  # need month to be represented with two digits so that 201212 (2012 December) is less than 2013(0)1 (2013 January)
            month = "0%s" % month
        return ("%s%s") % (year, month)

    time_low = ts_to_yearmonth(set_timestamp_low)
    time_high = ts_to_yearmonth(set_timestamp_high)

    return render_to_response("overview.html", {
        "statistics": json.dumps(statistics),
        "time_high": time_high,
        "time_low": time_low,
    }, context_instance=RequestContext(request))


def usertimeline(request, user_id):
    timeline_type = request.GET.get("linear", "no")
    linear_timeline = True if (timeline_type == "yes") else False
    timeline_start = request.GET.get("begin", "1999—20")
    timeline_end = request.GET.get("end", "2002—12")
    time_filter = request.GET.get("filter", "no")
    linear_timefilter = False if (time_filter == "no") else True
    user_parameters = {
        "timeline_type": timeline_type,
        "linear_timeline": linear_timeline,
        "time_filter": time_filter,
        "linear_timefilter": linear_timefilter,
        "timeline_start": timeline_start,
        "timeline_end": timeline_end,
    }

    from usertimeline import main
    meta_data_path = "%s%s.json" % (global_program_settings["meta_data_path"], user_id)
    with open(meta_data_path, "rb") as source:
        meta_data = json.load(source)

    return render_to_response("usertimeline.html", {
        "user_id": user_id,
        "graph_data": json.dumps(main(meta_data, user_parameters,)),
        "user_parameters": json.dumps(user_parameters),
    }, context_instance=RequestContext(request))


def usertod(request, user_id):
    meta_data_path = "%s%s.json" % (global_program_settings["meta_data_path"], user_id)
    with open(meta_data_path, "rb") as source:
        meta_data = json.load(source)

    from usertod import main
    tod_data = main(meta_data)

    return render_to_response("usertod.html", {
        "user_id": user_id,
        "times": json.dumps(tod_data),
    }, context_instance=RequestContext(request))


def usercom(request, user_id, direction):
    from usercom import main
    meta_data_path = "%s%s.json" % (global_program_settings["meta_data_path"], user_id)
    with open(meta_data_path, "rb") as source:
        meta_data = json.load(source)

    direction_text = "sent" if direction else "received"

    return render_to_response("usercom.html", {
        "user_id": user_id,
        "direction_text": direction_text,
        "graph_data": json.dumps(main(meta_data, direction)),
```

```
        "address_book_path": global_program_settings["address_book_static_path"],
    }, context_instance=RequestContext(request))


def messages(request, user_id, timestamp=False):
    from messages import main
    meta_data_path = "%s%s.json" % (global_program_settings["meta_data_path"], user_id)
    with open(meta_data_path, "rb") as source:
        meta_data = json.load(source)

    read_messages = ReadStatus.objects.filter(user_id=user_id)
    read_ids = []
    for m in read_messages:
        read_ids.append(m.file_id)

    timeline_messages = CaseTimeline.objects.filter(user_id=user_id)
    timeline_ids = []
    for m in timeline_messages:
        timeline_ids.append(m.file_id)

    def id_from_name(name):
        for index, attribute in enumerate(meta_data['attributes']):
            if attribute[0] == name:
                return index
        return None

    return render_to_response("messages.html", {
        "user_id": user_id,
        "timestamp": timestamp,
        "messages_data": json.dumps(main(meta_data)),
        "all_folders": json.dumps(meta_data["attributes"][id_from_name("meta_folder")][1]),
        "address_book_path": global_program_settings["address_book_static_path"],
        "read_messages": read_ids,
        "timeline_messages": timeline_ids,
    }, context_instance=RequestContext(request))


def case_timeline(request):
    data = CaseTimeline.objects.all()

    return render_to_response("casetimeline.html", {
        "data": data,
    }, context_instance=RequestContext(request))


def mark_message(request, user_id, file_id):
    # should check if user_id is among the user list
    entry = ReadStatus(
        user_id=user_id,
        file_id=file_id,
    )
    try:
        entry.save()
        status = "success"
    except:
        status = "seen before"

    return HttpResponse(status)


@csrf_exempt
def add_casetimeline(request):
    # Do not trust this input!
    user_id = request.POST.get("userid", None)
    file_id = request.POST.get("fileid", None)
    comment = request.POST.get("comment", "")
    timestamp = request.POST.get("timestamp", None)

    if timestamp:

        entry = CaseTimeline(
            user_id=user_id,
            file_id=file_id,
            comment=comment,
            timestamp=timestamp,
        )

        try:
            entry.save()
            status = "success"
        except:
            status = "added before"

    else:
        status = "timestamp missing"

    return HttpResponse(status)
```

E.13 server/models.py

```
from django.db import models


class ReadStatus(models.Model):
    user_id = models.CharField(max_length=100)
    file_id = models.IntegerField()

    class Meta:
            unique_together = (("user_id", "file_id"),)


class CaseTimeline(models.Model):
    user_id = models.CharField(max_length=100)
    file_id = models.IntegerField()
    comment = models.TextField(blank=True)
    timestamp = models.IntegerField()

    class Meta:
            unique_together = (("user_id", "file_id"),)
```

E.14 server/overview.py

```
# built in libraries in python 2.7.5
from multiprocessing import Pool, freeze_support, cpu_count
from collections import Counter
import datetime
import json
import os

# runtime variables
NUM_PROCESSES = cpu_count()
from server.settings import global_program_settings as global_settings
INDENT_JSON = global_settings["indent_json"]
META_DATA_PATH = global_settings["meta_data_path"]
CANDIDATE_MESSAGES_PATH = global_settings["candidate_messages_path"]
OVERVIEW_STATISTICS_PATH = global_settings["overview_statistics_path"]


def user_statistics(feature_file_path):
    with open(feature_file_path, "rb") as source:
        input_data = json.load(source)

    def id_from_name(name):
        for index, attribute in enumerate(input_data['attributes']):
            if attribute[0] == name:
                return index
        return None

    SENT_ID = id_from_name("email_sent")
    INTERNAL_ID = id_from_name("internal_receivers")
    EXTERNAL_ID = id_from_name("external_receivers")
    FORWARD_ID = id_from_name("email_subject_forward")
    REPLY_ID = id_from_name("email_subject_reply")
    SUSPECT_ID = id_from_name("other_party_suspect")
    SENDER_ID = id_from_name("email_sender")
    RECEIVER_ID = id_from_name("email_receivers")
    TIMESTAMP_LOCAL_ID = id_from_name("timestamp_local")
    EMAIL_MISSING_ID = id_from_name("email_missing")

    # various counter
    messages_sent = 0
    messages_received = 0
    dm_sent = 0
    dm_received = 0
    sent_fw = 0
    sent_re = 0
    suspect_sent_count = 0
    suspect_receive_count = 0
    external_sent_count = 0
    external_receive_count = 0
    unique_addresses_sent_to = []
    unique_addresses_received_from = []
    monthly_histogram_sent = Counter()
    monthly_histogram_received = Counter()
    weekday_histogram = Counter()
    timeofday_histogram = Counter()
    email_missing_count = 0

    timestamp_low = None
    timestamp_high = None

    for message in input_data['data']:
        recipients = message[INTERNAL_ID] + message[EXTERNAL_ID]

        timestamp = message[TIMESTAMP_LOCAL_ID]

        # determine range
        if timestamp < timestamp_low or timestamp_low is None:
            timestamp_low = timestamp
        if timestamp > timestamp_high or timestamp_high is None:
            timestamp_high = timestamp
```

```
        message_date = datetime.datetime.fromtimestamp(timestamp)
        year = message_date.year
        month = message_date.month
        if month < 10:  # need month to be represented with two digits so that 201212 (2012 December) is less than 2013(0)1 (2013 January)
            month = "0%s" % month
        unique_month = "%s%s" % (year, month)

        if message[SENT_ID]:
            messages_sent += 1
            if recipients == 1:
                dm_sent += 1
            if message[FORWARD_ID]:
                sent_fw += 1
            if message[REPLY_ID]:
                sent_re += 1
            if message[SUSPECT_ID]:
                suspect_sent_count += 1
            if message[EXTERNAL_ID] > 0:
                external_sent_count += 1
            for address in message[RECEIVER_ID]:
                if address not in unique_addresses_sent_to:
                    unique_addresses_sent_to.append(address)

            monthly_histogram_sent[unique_month] += 1
            weekday_histogram[message_date.weekday()] += 1
            timeofday_histogram[message_date.hour] += 1

        else:
            messages_received += 1
            if recipients == 1:
                dm_received += 1
            if message[SUSPECT_ID]:
                suspect_receive_count += 1
            if message[EXTERNAL_ID] > 0:
                external_receive_count += 1
            if message[SENDER_ID] not in unique_addresses_received_from:
                unique_addresses_received_from.append(message[SENDER_ID])

            monthly_histogram_received[unique_month] += 1

            # there are no mismatch in sent messages by definition
            email_missing_flag = message[EMAIL_MISSING_ID]
            if email_missing_flag:
                email_missing_count += 1

    return {
        "username": input_data['username'],
        "messages_sent": messages_sent,
        "messages_received": messages_received,
        "address": input_data['e-mail'],
        "duplicates": input_data['duplicate_messages'],
        "putback": input_data['put_back'],
        "dmSent": dm_sent,
        "dmRece": dm_received,
        "numFolders": len(input_data['attributes'][1][1]),
        "fwdSent": sent_fw,
        "replySent": sent_re,
        "suspSent": suspect_sent_count,
        "suspRece": suspect_receive_count,
        "extSent": external_sent_count,
        "extRece": external_receive_count,
        "addrTo": len(unique_addresses_sent_to),
        "addrFrom": len(unique_addresses_received_from),
        "numOriginal": input_data['original_messages'],
        "monthly_sent": monthly_histogram_sent,
        "monthly_rece": monthly_histogram_received,
        "weekday": weekday_histogram,
        "tod": timeofday_histogram,
        "timestamp_low": timestamp_low,
        "timestamp_high": timestamp_high,
        "email_missing": email_missing_count,
    }


### speed up code
def main():
    freeze_support()
    work_queue = []
    for (dirpath, dirnames, filenames) in os.walk(META_DATA_PATH):
        for filename in filenames:
            if filename.split(".")[-1] == "json":
                feature_file_path = ("%s%s") % (dirpath, filename)
                work_queue.append(feature_file_path)

    user_overview = []
    pool = Pool(processes=NUM_PROCESSES)
    for result in pool.imap_unordered(user_statistics, work_queue):
        user_overview.append(result)
    pool.close()
    pool.join()

    return user_overview  # order will be somewhat random depending on execution time of the processes
```

E.15 server/usertimeline.py

```
def main(meta_data, user_parameters):
    from collections import Counter
    import datetime
    import calendar
    import operator

    def id_from_name(name):
        for index, attribute in enumerate(meta_data['attributes']):
            if attribute[0] == name:
                return index
        return None

    EMAIL_FROM_ID = id_from_name("email_sender")
    EMAIL_TO_ID = id_from_name("email_receivers")
    DIRECTION_ID = id_from_name("email_sent")
    TIMESTAMP_LOCAL_ID = id_from_name("timestamp_local")

    cnt_day = Counter()
    ppl_day_storage = {}
    cnt_ppl_day_sent = []
    cnt_ppl_day_rece = []

    previous_message_day = None
    last_message_id = len(meta_data['data']) - 1  # -1: make it zero indexed

    for message_id, message in enumerate(meta_data['data']):
        email_sent = message[DIRECTION_ID]

        """ Determine number of messages per day and week in both directions.
        The combination year, week and day + direction will be unique for each day and week """
        timestamp = int(message[TIMESTAMP_LOCAL_ID])
        date = datetime.datetime.fromtimestamp(timestamp)
        year, week, day = date.isocalendar()
        week_str = ("%s") % week if (week > 9) else (("0%s") % week)

        unique_day = ("%s-%s-%s-%s") % (year, week_str, day, email_sent)
        cnt_day[unique_day] += 1

        """ Determine number of unique people talked to """
        def count_and_save(previous_message_day, cnt_ppl_day_sent, cnt_ppl_day_rece):
            year, week, day = previous_message_day
            week_str = ("%s") % week if (week > 9) else (("0%s") % week)
            unique_last_day_sent = ("%s-%s-%s-%s") % (year, week_str, day, True)
            unique_last_day_rece = ("%s-%s-%s-%s") % (year, week_str, day, False)
            ppl_day_storage[unique_last_day_sent] = len(set(cnt_ppl_day_sent))  # length unique
            ppl_day_storage[unique_last_day_rece] = len(set(cnt_ppl_day_rece))  # length unique

        current_day = (year, week, day)

        if previous_message_day and (current_day != previous_message_day):
            count_and_save(previous_message_day, cnt_ppl_day_sent, cnt_ppl_day_rece)
            cnt_ppl_day_sent = []  # reset counter
            cnt_ppl_day_rece = []  # reset counter

        if email_sent:  # add email addresses
            for addr in message[EMAIL_TO_ID]:  # can be many recipients
                cnt_ppl_day_sent.append(addr)
        else:  # received...
            cnt_ppl_day_rece.append(message[EMAIL_FROM_ID])  # can only be one sender

        if message_id == last_message_id:  # the store function is only called on week change, so this is a special case
            count_and_save(current_day, cnt_ppl_day_sent, cnt_ppl_day_rece)

        previous_message_day = current_day  # update the year and week for next round (last thing to happen)

    def year_week_to_timestamp(year, week, day=1):
        """ Functions for turning a (year, week, weekday) tuple into a timestamp
        Source of inspiration: from http://stackoverflow.com/questions/304256/whats-the-best-way-to-find-the-inverse-of-datetime-
            isocalendar """
        "The gregorian calendar date of the first day of the given ISO year"
        fourth_jan = datetime.date(year, 1, 4)
        delta = datetime.timedelta(fourth_jan.isoweekday() - 1)
        "Gregorian calendar date for the given ISO year, week and day"
        year_start = fourth_jan - delta
        date = year_start + datetime.timedelta(days=day - 1, weeks=week - 1)
        d = datetime.date(date.year, date.month, date.day)
        return calendar.timegm(d.timetuple())

    def weekday_series_graph(sent, data, file_name):
        day_names = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
        graph_json_data = []
        separated_days = {}
        unique_weekts = []

        earliest_date_year = int(data[0][0].split("-")[0])
        earliest_date_week = int(data[0][0].split("-")[1])
        latest_date_year = int(data[-1][0].split("-")[0])
        latest_date_week = int(data[-1][0].split("-")[1])
        linear_timeline = user_parameters["linear_timeline"]
        linear_timefilter = user_parameters["linear_timefilter"]

        # note: these calculations should preferable be done in JavaScript client side
        if linear_timefilter:
            earliest_date_year = int(user_parameters["timeline_start"][0:4])
            earliest_date_week = int(user_parameters["timeline_start"][5:7])
```

```
        latest_date_year = int(user_parameters["timeline_end"][0:4])
        latest_date_week = int(user_parameters["timeline_end"][5:7])

    earliest_timestamp = year_week_to_timestamp(earliest_date_year, earliest_date_week)
    latest_timestamp = year_week_to_timestamp(latest_date_year, latest_date_week)

    for day in range(1, 8):  # up to 8 means including 7
        separated_days[day] = []  # results in 7 empty lists: {1: [], 2: [], 3: [], 4: [], 5: [], 6: [], 7: []}

    print earliest_date_year, earliest_date_week

    if linear_timeline:
        year = earliest_date_year
        week = earliest_date_week
        weeks_current_year = int(datetime.date(year, 12, 31).strftime("%W"))
        while True:
            timestamp = year_week_to_timestamp(int(year), int(week))
            unique_weekts.append(timestamp)

            week += 1
            if week > weeks_current_year:
                week = 1
                year += 1
                weeks_current_year = int(datetime.date(year, 12, 31).strftime("%W"))

            if year == latest_date_year and week == latest_date_week:
                timestamp = year_week_to_timestamp(int(year), int(week))
                unique_weekts.append(timestamp)
                break

    for date, value in data:
        """  Convert to timestamp and append it to the correct day of week"""
        year, week, day, direction = date.split("—")
        timestamp = year_week_to_timestamp(int(year), int(week))  # iso_day is treated as 1—indexed in iso_to_gregorian()

        if linear_timefilter:
            if timestamp < earliest_timestamp or timestamp > latest_timestamp:
                continue

        if not linear_timeline:
            unique_weekts.append(timestamp)  # in order for the graphs in both directions to have the same time axis
        if direction == sent:  # only care for the current direction
            separated_days[int(day)].append([timestamp, value])
    if not linear_timeline:
        unique_weekts = set(unique_weekts)  # remove duplicate values

    """ if a week exist in either send or receive, make sure it is added in any case """
    for day in separated_days:
        for week_ts in unique_weekts:
            if week_ts not in [ts for ts, value in separated_days[day]]:
                separated_days[day].append([week_ts, 0])

    for day in separated_days:
        separated_days[day] = sorted(separated_days[day])

    """ build the graph JSON: (key, value) pairs for each day """
    for day in separated_days:
        graph_json_data.append({"key": day_names[day — 1], "values": separated_days[day]})

    return graph_json_data

cnt_day = sorted(cnt_day.items(), key=lambda item: item[0])
ppl_day_storage = sorted(ppl_day_storage.iteritems(), key=operator.itemgetter(0))
graph_data = {}
graph_data["messages_day"] = {}
graph_data["messages_day"]["sent"] = weekday_series_graph("True", cnt_day, "messages")  # sent
graph_data["messages_day"]["received"] = weekday_series_graph("False", cnt_day, "messages")  # received
graph_data["peers_day"] = {}
graph_data["peers_day"]["sent"] = weekday_series_graph("True", ppl_day_storage, "peers")  # sent
graph_data["peers_day"]["received"] = weekday_series_graph("False", ppl_day_storage, "peers")  # received

return graph_data
```

E.16 server/usertod.py

```
def main(meta_data):
    import time

    def id_from_name(name):
        for index, attribute in enumerate(meta_data['attributes']):
            if attribute[0] == name:
                return index
        return None

    TIMESTAMP_LOCAL_ID = id_from_name("timestamp_local")
    DIRECTION_ID = id_from_name("email_sent")

    times = []
    days = []
    current_dayofyear = None
    previous_timestamp = None

    def add_to_days(timestamp):
        timestamp_beginning_day = timestamp - (timestamp % 86400)  # adjust to the beginning of that day
        day_of_week = time.gmtime(timestamp).tm_wday  # 0 indexed
        days.append({"day": timestamp_beginning_day, "times": times, "weekday": day_of_week})  # modulo removed seconds since last new day

    for message in meta_data["data"]:

        if message[DIRECTION_ID]:  # it was sent

            m_timestamp = message[TIMESTAMP_LOCAL_ID]
            m_date = time.gmtime(m_timestamp)
            m_timeofday = round((((m_date.tm_hour * 60) + m_date.tm_min) / 60.0), 2)  # automatic float conversion
            m_dayofyear = m_date.tm_yday

            # if a new day is observed, add the current times to a new day in days and reset times
            if m_dayofyear > current_dayofyear and current_dayofyear is not None:
                add_to_days(previous_timestamp)
                times = []  # reset

            current_dayofyear = m_dayofyear

            # add this time of day to the current times array
            times.append(m_timeofday)
            previous_timestamp = m_timestamp

    if len(times) > 0:
        add_to_days(previous_timestamp)

    return days
```

E.17 server/usercom.py

```
from settings import global_program_settings
from datetime import datetime
from collections import Counter
from re import search

from pandas import DataFrame
import pandas as pd


def main(meta_data, direction):

    def id_from_name(name):
        for index, attribute in enumerate(meta_data['attributes']):
            if attribute[0] == name:
                return index
        return None

    EMAIL_FROM_ID = id_from_name("email_sender")
    EMAIL_TO_ID = id_from_name("email_receivers")
    DIRECTION_ID = id_from_name("email_sent")
    DATE_ID = id_from_name("timestamp_utc")
    COMPANY_DOMAIN = global_program_settings["company_domain"]
    all_addresses = []

    for message_id, message in enumerate(meta_data['data']):
        email_sent = message[DIRECTION_ID]
        email_date = message[DATE_ID]

        """ determine top and bottom peer communication """
        date = pd.Timestamp(datetime.fromtimestamp(email_date))
        if email_sent:
            for address in message[EMAIL_TO_ID]:
                all_addresses.append((email_sent, date, address))
        else:
            all_addresses.append((email_sent, date, message[EMAIL_FROM_ID]))

    # for each address find the number of messages to and from it
    df = DataFrame(all_addresses, columns=["direction", "date", "address"])
    stat = df.groupby(["address", "direction"]).count()
    # uncomment the drop statement on MAC.. weird differences between versions of Panda
    stat = stat.drop(["address", "direction"], axis=1)
    stat = stat.unstack().fillna(0)
```

```
    lstat = list(stat.itertuples())
    if direction:  # direction is boolean
        sort_key = 2
    else:
        sort_key = 1
    lstat = sorted(lstat, key=lambda key: key[sort_key], reverse=True)  # 0 is address, 1 is received, 2 is sent

    lstat = [entry for entry in lstat if entry[sort_key] > 0]

    def get_dates(addr):
        udf = df[df.address == addr].drop(["address"], axis=1)
        udf = udf.set_index("date")

        rval = {}
        for direction in ["sent", "received"]:  # True is sent, False is received

            direction_value = True if direction == "sent" else False

            udf_dir = udf[udf.direction == direction_value].drop(["direction"], axis=1)  # using 'is' won't work
            dates = list(udf_dir.itertuples())

            ctr = Counter()
            datevalues = {}
            for date in dates:
                yearweek = "%s-%s" % (date[0].year, date[0].week)
                ctr[yearweek] += 1
            for date, value in ctr.iteritems():
                datevalues[date] = value
            rval[direction] = datevalues

        return rval

    internal = []
    external = []
    for address, received, sent in lstat:
        if search(COMPANY_DOMAIN, address):  # regular expression search
            internal.append({"address": address, "sent": sent, "received": received, "dates": get_dates(address)})
        else:
            external.append({"address": address, "sent": sent, "received": received, "dates": get_dates(address)})

    result = {
        "internal": internal,
        "external": external,
    }

    return result
```

E.18 server/messages.py

```
def main(meta_data):

    def id_from_name(name):
        for index, attribute in enumerate(meta_data['attributes']):
            if attribute[0] == name:
                return index
        return None

    # input file mappings (used for raw visual UI)
    EMAIL_USERNAME_ID = id_from_name("meta_username")  # the username of the email (can be different than current user if added back)
    EMAIL_FROM_ID = id_from_name("email_sender")
    EMAIL_TO_ID = id_from_name("email_receivers")
    DIRECTION_ID = id_from_name("email_sent")
    TIMESTAMP_LOCAL_ID = id_from_name("timestamp_local")
    META_FOLDER_ID = id_from_name("meta_folder")
    META_MESSAGE_ID = id_from_name("meta_messageid")
    TIMESTAMP_UTC_ID = id_from_name("timestamp_utc")
    EMAIL_SUBJECT_ID = id_from_name("email_subject")
    PAYLOAD_LEN_ID = id_from_name("email_payload_len")
    NUM_INTERNAL_ID = id_from_name("internal_receivers")
    NUM_EXTERNAL_ID = id_from_name("external_receivers")
    DRAFT_ID = id_from_name("draft")
    OTHER_PARTY_SUSPECT_ID = id_from_name("other_party_suspect")
    MESSAGES_DAY_ID = id_from_name("messages_day")  # next in line
    MESSAGES_WEEK_ID = id_from_name("messages_week")
    UNIQUE_PPL_ID = id_from_name("unique_people_day")
    MESSAGE_ADDED_ID = id_from_name("added_back")
    FORWARD_ID = id_from_name("email_subject_forward")
    REPLY_ID = id_from_name("email_subject_reply")
    TIMEZONE_ID = id_from_name("timezone")
    ATTACHMENTS_ID = id_from_name("attachments")
    EMAIL_FOLDERS_ID = id_from_name("email_folders")
    EMAIL_MISSING_ID = id_from_name("email_missing")

    raw_messages_data = []
    for message in meta_data['data']:

        if not message[DIRECTION_ID]:  # received
            EMAIL_OTHER_PARTY = message[EMAIL_FROM_ID]
            NUM_OTHERS = 0
        else:  # sent by user
            NUM_OTHERS = len(message[EMAIL_TO_ID]) - 1
            try:
                EMAIL_OTHER_PARTY = message[EMAIL_TO_ID][0]  # the first recipient
            except:
```

```
                EMAIL_OTHER_PARTY = ""

            message_data = [
                message[EMAIL_USERNAME_ID],                 # 0
                message[META_FOLDER_ID],                # 1 remove
                message[META_MESSAGE_ID],               # 2 remove (add reference)
                message[DIRECTION_ID],                      # 3
                message[TIMESTAMP_LOCAL_ID],                # 4
                message[TIMESTAMP_UTC_ID],              # 5 # remove?
                message[EMAIL_SUBJECT_ID],                  # 6
                EMAIL_OTHER_PARTY,                      # 7 add +X
                message[PAYLOAD_LEN_ID],                    # 8
                message[NUM_INTERNAL_ID],                   # 9
                message[NUM_EXTERNAL_ID],                   # 10
                message[UNIQUE_PPL_ID],                 # 11 change name?
                message[MESSAGES_DAY_ID],               # 12
                message[MESSAGES_WEEK_ID],              # 13
                message[DRAFT_ID],                          # 14
                message[OTHER_PARTY_SUSPECT_ID],            # 15
                message[MESSAGE_ADDED_ID],                  # 16
                message[FORWARD_ID],                        # 17
                message[REPLY_ID],                          # 18
                NUM_OTHERS,                             # 19
                message[id_from_name("content_id")],        # 20
                message[TIMEZONE_ID],                       # 21
                message[ATTACHMENTS_ID],                    # 22
                message[EMAIL_FOLDERS_ID],                  # 23
                message[EMAIL_MISSING_ID],                  # 24

            ]
            raw_messages_data.append(message_data)

        return raw_messages_data
```

E.19 server/templates/index.html

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=10; IE=9; IE=8; IE=7; IE=EDGE; chrome=1" />
        <title>Email Visualizations</title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta name="description" content="">
        <meta name="author" content="">
        <link rel="stylesheet" href="/static/nv.d3.css">
        <style>
            html, body {
                height: 100%;
            }
            body {
                margin: 0;
                padding: 0;
                font-size: 10px;
                font-family: "Helvetica";
                color: #333333;
            }
            a {
                font-size: bold;
            }
        </style>
        {% block html_css %}{% endblock %}
    </head>
    <body>
    {% block html_body %}{% endblock %}
    <script src="/static/d3.min.js"></script>
    <script src="/static/nv.d3.min.js"></script>
    <script src="/static/jquery-2.1.0.min.js"></script>
    {% block html_javascript %}{% endblock %}
    </body>
</html>
```

E.20 server/templates/overview.html

```
{% extends "index.html" %}

{% block html_css %}
<style>
    table {
        border: 1px solid #E9E9E9;
        border-collapse: collapse;
        width: 100%;
    }
    table th {
        border: 1px solid #E9E9E9;
        padding: 4px;
        font-size: 9px;
        background: #F9FCFF;
    }
```

```
    table td {
        border: 1px solid #E9E9E9;
        padding: 4px;
    }
    .pctbox {
        width: 100%;
    }
    .pctbox_internal {
        font-size: 9px;
        float: left;
    }
    .clickable {
        cursor: pointer;
    }
    .clickable_td {
        background: #FDF4E7;
    }
    .control_link {
        cursor: pointer;
    }
    a {
        font-weight: bold;
    }
</style>
{% endblock %}

{% block html_body %}

<div style="margin: 15px 10px;">
<h2>Overview of all email accounts</h2>
<p>All the email accounts with key statistics. Some columns can be clicked to open more detailed visualizations. These are marked with a
       darker table header. Change the histogram <a class="control_link" data-variable="histogramBinary">type</a>. <!-- and set it's <a
       class="control_link" data-variable="histogramFilter">time filter</a>. --></p>

<table id="content">
    <thead>
        <tr>
            <th width="100" class="clickable_td" style="overflow: auto;">E-mail</th>
            <th width="50">Folders</th>
            <th width="60">Original<br>messages</th>
            <th width="60">Duplicates</th>
            <th width="60">Restored</th>
            <th width="60">Mismatches</th>
            <th width="60">Current<br>messages</th>
            <th class="clickable_td" width="60">Weekday</th>
            <th>Histogram sent</th>
            <th>Histogram received</th>
            <th width="100" class="clickable_td" >TimeOfDay</th>
            <th width="60" class="clickable_td">Unique<br>to</th>
            <th width="100">Messages sent</th>
            <th width="80" class="clickable_td">Unique<br>from</th>
            <th width="80">Forwarded</th>
            <th width="80">Replies</th>
            <th width="80">DM<br>Sent</th>
            <th width="80">DM<br>Received</th>
            <th width="80">Suspects<br>Sent</th>
            <th width="80">Suspects<br>Received</th>
            <th width="80">External<br>Sent</th>
            <th width="80">External<br>Received</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td colspan="21" style="text-align: center;">Processing data...</td>
        </tr>
    </tbody>
</table>
</div>

{% endblock %}

{% block html_javascript %}
<script src="/static/jquery-2.1.0.min.js"></script> <!-- http://jquery.com/ -->
<script src="/static/jquery.sparkline.min.js"></script> <!-- http://omnipotent.net/jquery.sparkline/#s-about -->
<script src="/static/jquery.floatThead.min.js"></script> <!-- http://mkoryak.github.io/floatThead/ -->

<script type="text/javascript">

var all_users = {{ statistics|safe }}; // the data

// various settings
var color_received = [0, 163, 255];
var color_sent = [0, 180, 0];
var color_both = [255, 0, 0];
var color_gray = [220, 220, 220];
var color_active = "yellow";

var controls = {
    "histogramBinary": true, // the bars are either 1 or 0
    "histogramFilter": true, // limit the range of the graph
    "histogramFilterBegin": {{ time_low }}, // year(4) followed  by month (2)
    "histogramFilterEnd": {{ time_high }},
}

var http_get = read_url();
if (http_get["begin"] == undefined) {
    var newURL = "/?begin="+ controls.histogramFilterBegin +"&end="+ controls.histogramFilterEnd;
```

```
    history.pushState(null, null, newURL);
};

$(".control_link").each(function() {
    mark_active($(this), controls[$(this).data("variable")]);
});

$(".control_link").click(function() {
    var control = $(this).data("variable");
    controls[control] = !controls[control]; //invert
    myMain();
    mark_active($(this), controls[control]);
});

function mark_active(element, value) {
    if (value) {
        element.css("background-color", color_active);
    } else {
        element.css("background-color", "auto");
    }
};

// order of functions used
function myMain(){
    if (controls.histogramFilter) {
        if (http_get["begin"] != undefined){
            controls.histogramFilterBegin = parseInt(http_get["begin"]);
            controls.histogramFilterEnd = parseInt(http_get["end"]);
        }
    }
    insert_data();
    // activate coloring of numbers
    colorcode(".num_folders", color_both);
    colorcode(".original_msg", color_both);
    colorcode(".current_msg", color_both);
    colorcode(".sent_to", color_sent);
    colorcode(".received_from", color_received);
    // activate the graphs from the sparklines library
    activate_sparklines();
    // activate floating table head
    $('table').floatThead();

    $(".clickable").click(function() { // clickable links
        var username = $(this).parent().data("username");
        var target = $(this).data("link");
        window.open('/' + target + '/' + username + '/'); // open in a new tab
    });
};
myMain()

// various functions
function insert_data(){

    var target_element = $("#content tbody").empty();
    all_users.forEach(function(d){

        //see overview.py for the object names
        var numSent = d.messages_sent;
        var numReceived = d.messages_received;
        var numCurrent = d.numOriginal - d.duplicates + d.putback;

        var row = ('<tr data-username="' + d.username + '">');
        row += ('<td data-link="messages" class="clickable">' + d.address + '</td>'); //email
        row += ('<td><div class="num_folders">' + d.numFolders + '</div></td>'); //folders
        row += ('<td><div class="original_msg">' + d.numOriginal + '</div></td>'); //original messages
        row += ('<td style="width: 90px;">' + pctbar(d.duplicates, d.numOriginal, color_both, color_gray) + '</td>'); //duplicates
        row += ('<td style="width: 90px;">' + pctbar(d.putback, numCurrent, color_both, color_gray) + '</td>'); //restored
        row += ('<td style="width: 90px;">' + pctbar(d.email_missing, numReceived, color_received, color_gray) + '</td>'); //restored
        row += ('<td><div class="current_msg">' + numCurrent + '</div></td>'); //current messages

        row += ('<td data-link="usertimeline" class="clickable"><div class="sparkline_histogram">' + histogram(d.weekday, 7) + '</div></td
            '); //weekday
        row += ('<td><div class="sparkline_graph_sent">' + histogram_monthly(d.monthly_sent) + '</div></td>'); //histogram
        row += ('<td><div class="sparkline_graph_rece">' + histogram_monthly(d.monthly_rece) + '</div></td>'); //histogram
        row += ('<td data-link="usertod" class="clickable"><div class="sparkline_histogram">' + histogram(d.tod, 24) + '</div></td>'); //
            timeofday

        row += ('<td data-link="usercom_sent" class="clickable"><div class="sent_to">' + d.addrTo + '</div></td>'); //unique to
        row += ('<td style="width: 90px;">' + pctbar(numSent, numCurrent, color_sent, color_received) + '</td>'); //messages sent
        row += ('<td data-link="usercom_received" class="clickable"><div class="received_from">' + d.addrFrom + '</div></td>'); //unique
            from
        row += ('<td style="width: 90px;">' + pctbar(d.fwdSent, numSent, color_sent, color_gray) + '</td>'); //forwarded
        row += ('<td style="width: 90px;">' + pctbar(d.replySent, numSent, color_sent, color_gray) + '</td>'); //replies
        row += ('<td style="width: 65px;">' + pctbar(d.dmSent, numSent, color_sent, color_gray) + '</td>'); //DM sent
        row += ('<td style="width: 65px;">' + pctbar(d.dmRece, numReceived, color_received, color_gray) + '</td>'); //DM received
        row += ('<td style="width: 65px;">' + pctbar(d.suspSent, numSent, color_sent, color_gray) + '</td>'); //suspects sent
        row += ('<td style="width: 65px;">' + pctbar(d.suspRece, numReceived, color_received, color_gray) + '</td>'); //suspects received
        row += ('<td style="width: 65px;">' + pctbar(d.extSent, numSent, color_sent, color_gray) + '</td>'); //external sent
        row += ('<td style="width: 65px;">' + pctbar(d.extRece, numReceived, color_received, color_gray) + '</td>'); //external received
        row += ('</tr>');

        target_element.append(row); //add the row to the table
    });
};

function normalize(a, b) {
    return (a * 100 / b).toFixed(1);
```

```
};

function pctbar(num, all, color1, color2) {
    // show num as a percentage of all and color num with color1 and the remaining with color2
    var pct = normalize(num, all);
    var str = '<div class="pctbox">';
    str += '<div class="pctbox_internal" style="background: rgba('+color1[0]+','+color1[1]+','+color1[2]+','+0.46+'); width: '+ pct +'%;">
        '+ num +'</div>';
    str += '<div class="pctbox_internal" style="background: rgba('+color2[0]+','+color2[1]+','+color2[2]+','+0.46+'); width: '+ (100 - pct)
        .toFixed(1) +'%;"> </div>';
    str += '</div>';
    return str;
}

function histogram_monthly(data) {
    // Create an array with either the given value or 0 for every month between the given start and end range
    function year_month_split(number, start, end) {
        return parseInt(number.toString(10).substring(start, end));
    }

    var year = year_month_split(controls.histogramFilterBegin, 0, 4);
    var month = year_month_split(controls.histogramFilterBegin, 4, 6);
    var year_stop = year_month_split(controls.histogramFilterEnd, 0, 4);
    var month_stop = year_month_split(controls.histogramFilterEnd, 4, 6);

    month_stop += 1;
    if (month_stop == 13) {
        year_stop += 1;
        month_stop = 1;
    }
    var histogram_data = [];

    while (true) {
        if (month < 10) {
            var reference = year + "0" +  month;
        }
        else {
            var reference = year + "" +  month;
        }
        var month_count = data[reference];
        if (month_count != undefined) {
            if (controls.histogramBinary) {
                histogram_data.push(1);
            }
            else {
                histogram_data.push(month_count);
            }
        }
        else {
            histogram_data.push(0);
        }

        month += 1
        if (month == 13) {
            year += 1;
            month = 1;
        }
        if (year == year_stop && month == month_stop) {
            break;
        }
    }
    return histogram_data.join(",");
}

function histogram(data, range){
    // fill all missing values in data not in range with 0
    var histogram_data = [];
    for (i=0; i<range; i++) {
        var value = data[i];
        if (value != undefined) {
            histogram_data.push(value);
        }
        else {
            histogram_data.push(0);
        }
    }
    return histogram_data.join(",");
}

function colorcode(class_label, color) { // vary the color intensity as a function of value
    var maximum = 0;
    var elements = $(class_label);

    elements.each(function(){
        var value = parseInt($(this).html());
        if (value > maximum) {
            maximum = value;
        }
    });

    elements.each(function(){
        var pct = ($(this).html() / maximum).toFixed(3);
        $(this).css("background-color", "rgba("+color[0]+","+color[1]+","+color[2]+"," + pct + ")");
    });
}

function activate_sparklines(){ // activate two kinds of graph api's on the prepared data
```

```
    var bar_spacing = 0;
    if (controls.histogramFilter) {
        bar_spacing = 1;
    }

    $('.sparkline_graph_sent').sparkline('html', {
        type: 'line',
        width: 230,
        height: 12,
        lineColor: "rgb(0,0,0)",
        fillColor: "rgba("+color_sent[0]+","+color_sent[1]+","+color_sent[2]+",0.66)",
        spotColor: undefined,
        minSpotColor: undefined,
        maxSpotColor: undefined,
        highlightSpotColor: undefined,
        highlightLineColor: undefined,
        spotRadius: NaN,
        chartRangeMin: NaN,
        chartRangeMax: NaN,
        chartRangeMinX: NaN,
        chartRangeMaxX: NaN,
        normalRangeMin: NaN,
        normalRangeMax: NaN,
        normalRangeColor: '#e5e5e5',
        drawNormalOnTop: false,
        disableTooltips: true,
    });

    $('.sparkline_graph_rece').sparkline('html', {
        type: 'line',
        width: 230,
        height: 12,
        lineColor: "rgb(0,0,0)",
        fillColor: "rgba("+color_received[0]+","+color_received[1]+","+color_received[2]+",0.66)",
        spotColor: undefined,
        minSpotColor: undefined,
        maxSpotColor: undefined,
        highlightSpotColor: undefined,
        highlightLineColor: undefined,
        spotRadius: NaN,
        chartRangeMin: NaN,
        chartRangeMax: NaN,
        chartRangeMinX: NaN,
        chartRangeMaxX: NaN,
        normalRangeMin: NaN,
        normalRangeMax: NaN,
        normalRangeColor: '#e5e5e5',
        drawNormalOnTop: false,
        disableTooltips: true,
    });

    $('.sparkline_histogram').sparkline('html', {
        type: 'bar',
        barWidth: 2,
        barSpacing: 1,
        zeroAxis: false,
        barColor: "rgb("+color_sent[0]+","+color_sent[1]+","+color_sent[2]+")",
        disableTooltips: true,
    });
};

// function for reading GET data (URL)
function read_url(){
    var url = document.location.search;
    url = url.split("?")[1];
    variables = {};
    if (url != undefined) {
        parts = url.split("&");
        for (part_id in parts) {
            var part = parts[part_id].split("=");
            variables[decodeURIComponent(part[0])] = decodeURIComponent(part[1]);
        }
    }
    return variables
}

</script>
{% endblock %}
```

E.21 server/templates/usertimeline.html

```
{% extends "index.html" %}

{% block html_css %}
<style>
    .box svg {
        height: 200px;
    }
    .scrollbox{
        height: 250px;
        overflow: auto;
    }
    .nvd3 .nv-axis line {
        stroke: #ffffff;
```

```
    }
    #topbottom{
        width: 100%;
    }

</style>
{% endblock %}


{% block html_body %}

<div style="margin: 15px 10px;">
    <h2>Timeline for {{ user_id }}</h2>
    <p>Click on a time period to see the messages for that period. Toggle linear time by setting "linear=yes" in the address bar. Same goes
        for time filter: Set "filter=yes" and adjust the range in terms of "year-week". Use the weekday legend for filtering on day of
        week. Click in the graph to see the messages for that current day.</p>

    <p>Other views: <a href="{% url "usertimeline" user_id %}">timeline</a>, <a href="{% url "usertod" user_id %}">time of day</a>, <a href
        ="{% url "usercom_sent" user_id %}">peer view sent</a>, <a href="{% url "usercom_received" user_id %}">peer view received</a></p
        >

    <div class="box" id="messages_sent"><h4>Messages sent</h4><svg></svg></div>
    <div class="box" id="peers_sent"><h4>Unique peers sent to</h4><svg></svg></div>
    <div class="box" id="messages_received"><h4>Messages received</h4><svg></svg></div>
    <div class="box" id="peers_received"><h4>Unique peers received</h4><svg></svg></div>
</div>

{% endblock %}


{% block html_javascript %}
<script type="text/javascript">

var user_parameters = {{ user_parameters|safe }};
var graph_data = {{ graph_data|safe }};

var newURL = "?linear="+user_parameters.timeline_type+"&filter="+user_parameters.time_filter+"&begin="+user_parameters.timeline_start+"&end
    ="+user_parameters.timeline_end+"";
history.pushState(null, null, newURL);

var millisecPerDay = 86400000;
var secPerDay = 86400;

function timestampToString(timestamp){
    var a = new Date(timestamp);
    var months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'];
    var year = a.getFullYear();
    var month = months[a.getMonth()];
    var date = a.getDate();
    return(date + ' ' + month + ', ' + year);
}

var messages_sent_chart = newMultiBarChart(graph_data["messages_day"]["sent"], d3.select('#messages_sent svg'), "messages", true);
var messages_received_chart = newMultiBarChart(graph_data["messages_day"]["received"], d3.select('#messages_received svg'), "messages",
    true);
var peers_sent_chart = newMultiBarChart(graph_data["peers_day"]["sent"], d3.select('#peers_sent svg'), "peers", false);
var peers_received_chart = newMultiBarChart(graph_data["peers_day"]["received"], d3.select('#peers_received svg'), "peers", false);

var day_names = {"Mon": 0, "Tue": 1, "Wed": 2, "Thu": 3, "Fri": 4, "Sat": 5, "Sun": 6};

function newMultiBarChart(data, target, text, stacked) {
    var chart;
    nv.addGraph(function() {
        chart = nv.models.multiBarChart()
                .x(function(d) { return d[0] * 1000; })
                .y(function(d) { return d[1]; })
                .transitionDuration(100)
                .showControls(stacked)
                .color(['#DD9898', '#D07171', '#C34B4B', '#A13636', '#7B2828', '#68CE19', '#5AAD22'])
                .tooltipContent(function(key, x, y, e) {
                    var days_to_add = day_names[e.series.key]; // NOT e.point.series since it changes depending on what is viewed
                    var timestamp = e.point[0] * 1000;
                    timestamp += days_to_add * millisecPerDay;
                    var time_string = timestampToString(timestamp);
                    return '<h3>' + key + '</h3>' + '<p>' +  parseInt(y) + ' ' + text + '<br>' + time_string +  '</p>'
                });

        chart.multibar.stacked(stacked);

        //Format x-axis labels with custom function.
        chart.xAxis.tickFormat(function(d) {
            return d3.time.format('%e %b \'%y')(new Date(d))
        });
        chart.yAxis.tickFormat(d3.format(',.2f'));

        target.datum(data).call(chart);

        nv.utils.windowResize(chart.update);
        chart.multibar.dispatch.on("elementClick", function(e) {
            console.log(e)
            day_timestamp = e.point[0] + (day_names[e.series.key] * secPerDay);
            // django template code..
            window.open( '{% url 'messages' user_id  %}' + day_timestamp + '/' );
        });
    });
    return chart;
}
```

```
</script>
{% endblock %}
```

E.22 server/templates/usertod.html

```
{% extends "index.html" %}

{% block html_css %}
<style>
html{
}
#graph{
    height: 500px;
    width: 100%;
}
.axis path,
.axis line {
    fill: none;
    stroke: #DDDDDD;
    shape-rendering: crispEdges;
}
.normal_working_hours{
    fill: rgba(119, 225, 245, 0.07);
}
.axis text {
    font-family: sans-serif;
    font-size: 10px;
    color:blue;
}
svg circle,
.control_link {
    cursor: pointer;
}
a {
    font-weight: bold;
}
</style>
{% endblock %}


{% block html_body %}
<div style="margin: 15px 10px;">
    <h2>Time of day for {{user_id}}</h2>
    <p>This chart shows how <b>sent</b> messages are spread out during the day. Black is used for Monday to Friday and green for Saturday
        and Sundays.<br>You can change how <a class="control_link" data-variable="showLines">lines</a> and
    <a class="control_link" data-variable="showPoints">dots</a> are displayed, and you can change the way
    <a class="control_link" data-variable="linearTime">time</a> is rendered.</p>

    <p>Other views: <a href="{% url "usertimeline" user_id %}">timeline</a>, <a href="{% url "usertod" user_id %}">time of day</a>, <a href
        ="{% url "usercom_sent" user_id %}">peer view sent</a>, <a href="{% url "usercom_received" user_id %}">peer view received</a></p
        >

    <div id="graph"></div>
</div>

{% endblock %}


{% block html_javascript %}
<script type="text/javascript">

controls = {
    "showLines": true,
    "showPoints": true,
    "linearTime": true,
};

var workingHours = {"begin": 8, "end": 16};
var color_active = "yellow";
var color_working_day = "black";
var color_weekend_day = "green";

$(".control_link").each(function() {
    mark_active($(this), controls[$(this).data("variable")]);
});

$(".control_link").click(function() {
    var control = $(this).data("variable");
    controls[control] = !controls[control]; //invert
    draw();
    mark_active($(this), controls[control]);
});

function mark_active(element, value) {
    if (value) {
        element.css("background-color", color_active);
    } else {
        element.css("background-color", "auto");
    }
};

newData = {{ times|safe }};
//console.log(newData)
```

```
/*
// test data: uncomment in order to replace data for verification
newData = [
    {"day": 1262304000, "times": [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24], "weekday": 1},
    {"day": 1262400400, "times": [0,1,2,3,4,5,6,7,8,9,10,11,12], "weekday": 2},
    {"day": 1262500400, "times": [8,9,10,11,12,13,14,15,16], "weekday": 3},
    {"day": 1262600400, "times": [12,13], "weekday": 4},
    {"day": 1262700400, "times": [12], "weekday": 5},
    {"day": 1262800400, "times": [12,13,14,15,16,17,18,19,20,21,22,23,24], "weekday": 6},
    {"day": 1263304000, "times": [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24], "weekday": 1},
//times come in resolution of minutes with 2 decimal places i.e 3.50 is 3 hours and 30 minutes after midnight.
]
*/


//convert timestamps to Date objects
newData.forEach(function(d){
    d.day = new Date(d.day * 1000);
})

// draw the graph
var canvas = draw();

function draw(){
    return drawGraph("#graph", newData);
}

function drawGraph(target, data) {
    var graphWidth = $(target).width();
    var graphHeight = $(target).height();
    var padding = 25;
    var numDays = data.length;
    var barWidth = graphWidth  / (4*numDays);
    if (barWidth > 3) {barWidth = 3}
    var nonLinearIncrement = (graphWidth - (padding*2)) / numDays;
    var dateMin = d3.min(data, function(d){ return d.day; });
    var dateMax = d3.max(data, function(d){ return d.day; });

    var dateScale = d3.time.scale()
        .domain([dateMin, dateMax]) // map available dates evenly
        .range([(padding + barWidth) + barWidth, graphWidth - barWidth - padding]);
        // start x and end x. barWidth in order for the last bar not to disappear.

    var hourScale = d3.scale.linear()
        .domain([0, 24]) // hours a day
        .range([graphHeight - padding, padding]);

    d3.select("svg").remove(); // clear any old svg element

    var canvas = d3.select(target)
        .append("svg")

    // create the y-axis using hour of day information
    var yAxis = d3.svg.axis()
        .scale(hourScale)
        .orient("left")
        .ticks(24);

    canvas.append("g")
        .attr("class", "axis")
        .attr("transform", "translate("+padding+", 0)")
        .call(yAxis);

    // create the x-axis using date information
    if (controls.linearTime){
        var xAxis = d3.svg.axis()
            .scale(dateScale)
            .orient("bottom")
            .tickFormat(d3.time.format("%d.%b %y"));

        canvas.append("g")
            .attr("class", "axis")
            .attr("transform", "translate(0, "+ (graphHeight - padding) +")")
            .call(xAxis);
    }

    var fillFunction = function(d, i) {
        if (d.weekday in [5, 6]) {
            return color_working_day;
        }
        return color_weekend_day;
    }

    if (controls.showLines) {
        var nonLinearX = padding;
        var lines = canvas.selectAll("rect")
            .data(data)
            .enter()
            .append("rect")
            .attr({
                "x": function(d, i) {
                    if (controls.linearTime){
                        return dateScale(d.day);
                    }
                    else {
                        nonLinearX += nonLinearIncrement;
```

```
                    return nonLinearX;
                }
            },
            "y": function(d, i) { return hourScale( d3.max(d.times) ); },
            "height": function(d, i) {
                var max = d3.max(d.times);
                var min = d3.min(d.times);
                return hourScale( 24 - (max - min)) - padding;
            },
            "width": barWidth,
            "fill": fillFunction,
        });
    }
    // normal working hours
    canvas.append("rect")
        .attr({
            "x": padding,
            "y": hourScale(workingHours.end),
            "height": hourScale(24 - workingHours.begin) - padding,
            "width": graphWidth,
            "class": "normal_working_hours",
        });

    if (controls.showPoints) {
        var nonLinearX = padding;
        var lastDay = false;
        // add one point per time. the data structure must first be unpacked.
        var allTimes = [];
        data.forEach(function(day){
            day.times.forEach(function(time){
                allTimes.push({"day": day.day, "time": time, "weekday": day.weekday});
            });
        });
        var points = canvas.selectAll("circle")
            .data(allTimes)
            .enter()
            .append("circle")
            .attr({
                "cx": function(d, i) {
                    if (controls.linearTime){
                        return dateScale(d.day);
                    }
                    else {
                        if (lastDay != d.day){
                            nonLinearX += nonLinearIncrement;
                        }
                        lastDay = d.day;
                        return nonLinearX;
                    }
                },
                "cy": function(d, i) { return hourScale(d.time); },
                "r": barWidth * 2,
                "fill": fillFunction,
            })
            .on('click', function(d, i) {
                window.open( "/messages/{{user_id}}/"+ (+d.day / 1000) +"/" );
            });
    }

    return canvas;
}


</script>
{% endblock %}
```

E.23 server/templates/usercom.html

```
{% extends "index.html" %}

{% block html_css %}
<style>
    table {
        text-align: left;
        border: 1px solid #E9E9E9;
        border-collapse: collapse;
        width: 100%;
    }
    table th {
        border: 1px solid #E9E9E9;
        padding: 4px;
        font-size: 9px;
        background: #F9FCFF;
    }
    table td {
        border: 1px solid #E9E9E9;
        padding: 4px;
    }
    .address_box{
        width: 50%;
        float: left;
    }
    a {
        font-weight: bold;
```

```
    }
    .clickable {
        cursor: pointer;
    }

</style>
{% endblock %}


{% block html_body %}

<div style="margin: 15px 10px;">
    <h2>Individual communication patterns for {{user_id}}: {{direction_text}}</h2>

    <p>All email addresses with 1 or more messages in selected direction ({{direction_text}}). Change the histogram <a class="control_link"
            data-variable="histogram_binary">type</a>.</p>

    <p>Other views: <a href="{% url "usertimeline" user_id %}">timeline</a>, <a href="{% url "usertod" user_id %}">time of day</a>, <a href
           ="{% url "usercom_sent" user_id %}">peer view sent</a>, <a href="{% url "usercom_received" user_id %}">peer view received</a></p
           >

    <div class="address_box">
        <h4>Internal</h4>
        <table id="internal" width="100%">
            <thead></thead>
            <tbody><tr><td>Loading...</td></tr></tbody>
        </table>
    </div>
    <div class="address_box">
        <h4>External</h4>
        <table id="external" width="100%">
            <thead></thead>
            <tbody><tr><td>Loading...</td></tr></tbody>
        </table>
    </div>

</div>

{% endblock %}


{% block html_javascript %}
<script src="/static/jquery-2.1.0.min.js"></script>
<script src="/static/jquery.sparkline.min.js"></script>
<script src="/static/jquery.floatThead.min.js"></script>

<script type="text/javascript">
var graph_data = {{ graph_data|safe }};
var address_book_path = "{{ address_book_path }}";
var address_book;

var color_sent = "rgba(0, 180, 0, 0.76)";
var color_received = "rgba(0,163,255,0.66)";
var color_active = "yellow";

var options = {
    "histogram_binary": true,
    "histogram_filter_begin": "1999-01",    //(year-week): week is 1 indexed
    "histogram_filter_end": "2003-01",      //end is inclusive
}


var http_get = read_url();
if (http_get["begin"] == undefined) {
    var newURL = "?begin="+ options.histogram_filter_begin +"&end="+ options.histogram_filter_end;
    history.pushState(null, null, newURL);
};

$(".control_link").each(function() {
    mark_active($(this), options[$(this).data("variable")]);
});

$(".control_link").click(function() {
    var control = $(this).data("variable");
    options[control] = !options[control]; //invert
    myMain();
    mark_active($(this), options[control]);
});

function mark_active(element, value) {
    if (value) {
        element.css("background-color", color_active);
    } else {
        element.css("background-color", "auto");
    }
};

function myMain(){
    if (http_get["begin"] != undefined){
        options.histogram_filter_begin = http_get["begin"];
        options.histogram_filter_end = http_get["end"];
    }
    getAddressBook();
}
myMain();

function getAddressBook() {
```

```
    $.get(address_book_path, function(respons) {
        address_book = respons;
        populate_topbottom();
    });
}

function populate_topbottom() {
    // receive a structure divided in internal and external.
    // each item has an email, messages sent and messages received from that address and an array of dates
    // the dates have "year-week" as index and a value of how many messages were observed that week


    for (type_id in graph_data) {
        var thead_content = ["Email address", "Sent", "Received", "Timeline"];
        var thead = $("<tr></tr>");
        for (idx in thead_content) {
            thead.append("<th>" + thead_content[idx] + "</th>");
        }
        var type = graph_data[type_id];

        var tbody = "";
        for (line_id in type) {
            line = type[line_id];
            tbody += ("<tr>");
            tbody += ("<td>" + emailToLink(line.address) + "</td>");
            tbody += ("<td width='35px'>" + line.sent + "</td>");
            tbody += ("<td width='35px'>" + line.received +"</td>");
            tbody += ("<td class='clickable' data-address='"+line.address+"'><div class='sparkline_graph'>" + histogram_monthly(line.dates.
                   sent, "sent") + "</div>")
            tbody += ("<div class='sparkline_graph'>" + histogram_monthly(line.dates.received, "received") + "</div></td>")
            tbody += ("</tr>");

        }
        $("#" + type_id + " thead").empty().append(thead)
        $("#" + type_id + " tbody").empty().html(tbody)
    }

    activate_sparklines();

    $(".clickable").click(function(){
        window.open("/messages/{{user_id}}/?filter_address=" + $(this).data("address"));

    })
}

function emailToLink(email) {
    if (address_book[email]) {
        var username = address_book[email];
        var path = "/messages/" + username + "/";
        return "<a href=\"" + path + "\" target=\"_blank\">" + email + "</a>";
    }
    else {
        return email;
    }
}

function activate_sparklines(){
    var bar_spacing = 0;
    if (options["histogram_filter"]) {
        bar_spacing = 1;
    }

    $('.sparkline_graph').sparkline('html', {
        type: 'bar',
        barWidth: 2,
        barSpacing: 2,
        height: 12,
        barColor: color_sent,
        negBarColor: color_received,
        disableTooltips: true,
    });
}

//copied and reused from overview.html
function histogram_monthly(data, type) {
    var year = parseInt(options["histogram_filter_begin"].split("-")[0]);
    var week = parseInt(options["histogram_filter_begin"].split("-")[1]);
    var year_stop = parseInt(options["histogram_filter_end"].split("-")[0]);
    var week_stop = parseInt(options["histogram_filter_end"].split("-")[1]);

    var binary_default = 1;
    if (type == "received"){
        binary_default = -1;
    }

    //stop one week later
    week_stop += 1;
    if (week_stop == 53) {
        year_stop += 1;
        week_stop = 1;
    }
    var histogram_data = [];

    while (true) {
        var reference = year + "-" + week;
        var week_count = data[reference];
        if (week_count != undefined) {
```

```
            if (options["histogram_binary"]) {
                histogram_data.push(binary_default);
            }
            else {
                if (type == "received"){
                    histogram_data.push(-week_count);
                }
                else {
                    histogram_data.push(week_count);
                }
            }
        }
        else {
            histogram_data.push(0);
        }

        week += 1
        if (week == 53) {
            year += 1;
            week = 1;
        }
        if (year == year_stop && week == week_stop) {
            break;
        }
    }
    return histogram_data.join(",");
}

// function for reading GET data (URL)
function read_url(){
    var url = document.location.search;
    url = url.split("?")[1];
    variables = {};
    if (url != undefined) {
        parts = url.split("&");
        for (part_id in parts) {
            var part = parts[part_id].split("=");
            variables[decodeURIComponent(part[0])] = decodeURIComponent(part[1]);
        }
    }
    return variables
}

</script>
{% endblock %}
```

E.24 server/templates/messages.html

```
{% extends "index.html" %}

{% block html_css %}
<style>
    table {
        width: 100%;
        border: 1px solid #E9E9E9;
        border-collapse: collapse;
        border-spacing: 0;
    }
    table th {
        border: 1px solid #E9E9E9;
        padding: 5px;
        font-size: 9px;
        background: #F9FCFF;
    }
    table td {
        border: 1px solid rgb(206, 206, 206);
        padding: 2px;
    }
    #main_top {
        width: 100%;
        height: 60%;
        overflow: auto;
    }
    #main_bottom {
        width: 100%;
        height: 39%;
        overflow: auto;
        border-top: 3px solid rgb(206, 206, 206);
    }
    table tbody tr:hover {
        cursor: pointer;
    }
    .detail_box{
        float: left;
        height: 100%;
        overflow: auto;
    }
    a {
        color: #0016B4;
        cursor: pointer;
    }
    a.link_active{
        background-color: rgb(236, 227, 0);
    }
```

```
</style>
{% endblock %}

{% block html_body %}
<div id="main_top">
    <div id="statistics">
        <div id="links">
            <a class="filter_activators" data-variable="onlySent">Sent only</a> |
            <a class="filter_activators" data-variable="onlyReceived">Received only</a> |
            <a class="filter_activators" data-variable="onlyDM">Direct Messages only</a> |
            <a class="filter_activators" data-variable="mass_email_filter">Mass email only</a> |

        </div>
        <div id="pagination"></div>
    </div>
    <table id="table" width="100%">
        <thead style="background: rgb(255, 226, 203);">
            <tr style="text-align: center;">
                <th><a href="{% url "case_timeline" %}">CTL</a></th>
                <th>{% if timestamp %}<a class="link_active" href="{% url 'messages' user_id %}">Date reset</a>{% else %}Date{% endif %}</
                    th>
                <th>Day</th>
                <th>Time</th>
                <th style="text-align: left;">The other</th>
                <th><a class="filter_activators" data-variable="haveInternal">Int</a></th>
                <th><a class="filter_activators" data-variable="haveExternal">Ext</a></th>
                <th>Dir</th>
                <th style="text-align: left;">Subject</th>
                <th><a class="filter_activators" data-variable="hasAttachments">Attachments</a></th>
                <th><a class="filter_activators" data-variable="onlyFW">Fw</a></th>
                <th><a class="filter_activators" data-variable="onlyRE">Re</a></th>
                <th>Length</th>
                <th><a class="filter_activators" data-variable="suspects_only_filter">Suspects</a></th>
                <th><a class="filter_activators" data-variable="msg_added_filter">Deleted</a></th>
                <th><a class="filter_activators" data-variable="isDraft">Draft</a></th>
                <th><a class="filter_activators" data-variable="isMismatch">Mismatch</a></th>
                <th width="35">#Day</th>
                <th width="35">#Week</th>
                <th width="35">#Peers/week</th>
                <th>Folder</th>
            </tr>
        </thead>
        <tbody></tbody>
    </table>
</div>
<div id="main_bottom">
    <div id="personal_folders" class="detail_box" style="width: 10%;"><b>Account folders</b><br></div>
    <div class="detail_box" style="width: 20%;">
        <div style="margin: 15px"><b style="font-size: 12pt;">Headers preview</b><br> <span id="header_box_details">Details</span><hr></div
            >
        <div style="margin: 15px" id="header_box"></div>
    </div>
    <div class="detail_box" style="width: 30%; font-size: 12px;">
        <div style="margin: 15px"><b style="font-size: 12pt;">Text/plain preview</b><br> <span id="plain_box_details">Details</span><hr></
            div>
        <div style="margin: 15px" id="plain_box"></div>
    </div>
    <div class="detail_box" style="width: 10%;">
        <div style="margin: 15px"><b style="font-size: 12pt;">Attachment preview</b><br> <span id="attachment_box_details">Details</span><
            hr></div>
        <div style="margin: 15px" id="attachment_box"></div>
    </div>
    <div class="detail_box" style="width: 29%;">
        <div style="margin: 15px"><b style="font-size: 12pt;">Text/HTML preview</b><br> <span id="html_box_details">Details</span><hr></div
            >
        <iframe style="text-align: center; border: 0px; width: 100%; height: 80%;" id="html_box_iframe" src="about:blank"></iframe>
    </div>
</div>
{% endblock %}


{% block html_javascript %}
<script src="/static/date.format.js"></script>
<script type="text/javascript">

var EMAIL_USERNAME_ID    = 0;
var META_FOLDER_ID       = 1;
var META_MESSAGE_ID      = 2;
var DIRECTION_ID         = 3;
var TIMESTAMP_LOCAL_ID   = 4;
var TIMESTAMP_UTC_ID     = 5;
var EMAIL_SUBJECT_ID     = 6;
var EMAIL_OTHER_PARTY_ID = 7;
var PAYLOAD_LEN_ID       = 8;
var NUM_INTERNAL_ID      = 9;
var NUM_EXTERNAL_ID      = 10;
var UNIQUE_PPL_ID        = 11;
var MESSAGES_DAY_ID      = 12;
var MESSAGES_WEEK_ID     = 13;
var DRAFT_ID             = 14;
var TO_FROM_SUSPTECT_ID = 15;
var MESSAGE_ADDED_ID     = 16;
var FORWARD_ID           = 17;
var REPLY_ID             = 18;
var NUM_OTHERS           = 19;
```

```
var CONTENT_ID            = 20;
var TIMEZONE_ID           = 21;
var ATTACHMENT_ID         = 22;
var EMAIL_FOLDERS_ID      = 23;
var EMAIL_MISSING_ID      = 24;

var filter_settings = {
    "timestamp_filter": {% if timestamp %}true{% else %}false{% endif %},
    "timestamp_start": {{timestamp|default:"false"}},
    "timestamp_duration": 7, // days
    "word_filter": false,
    "word_keyword": "Update",
    "msg_added_filter": false,
    "mass_email_filter": false,
    "mass_email_filter_limit": 10,
    "suspects_only_filter": false,
    "haveInternal": false,
    "haveExternal": false,
    "onlySent": false,
    "onlyReceived": false,
    "onlyDM": false,
    "hard_display_limit": 750,
    "onlyFW": false,
    "onlyRE": false,
    "notREFW": false,
    "timestamp_choice": TIMESTAMP_LOCAL_ID,
    "isDraft": false,
    "isMismatch": false,
    "hasAttachments": false,
};

var color_read = "rgb(231, 231, 231)";
var color_active = "rgb(236, 227, 0)";
var color_ctl = "rgb(189, 234, 255)";

function color_code_row(row, id){
    if (timeline_messages.indexOf(id) != -1) {
        row.css({
            "background": color_ctl,
        })
    } else if (read_messages.indexOf(id) != -1) {
        row.css({
            "background": color_read,
        });
    }
}

function mark_active(element, value) {
    if (value) {
        element.addClass("link_active");
    } else {
        element.removeClass("link_active");
    }
};

$(".filter_activators").each(function() {
    mark_active($(this), filter_settings[$(this).data("variable")]);
});

$(".filter_activators").click(function() {
    var control = $(this).data("variable");
    filter_settings[control] = !filter_settings[control]; //invert
    reset_start_index(); // reset position where to start displaying after filtering
    filter_metadata();
    mark_active($(this), filter_settings[control]);
});

var address_book_path = "{{ address_book_path }}";
var user_id = "{{ user_id }}";
var read_messages = {{ read_messages }};
var timeline_messages = {{ timeline_messages }};



//runtime variables
var address_book;
var all_messages = {{ messages_data|safe }};
var filtered_messages;
var lastViewed = null; //the last active raw message
var display_start_index, display_end_index;
reset_start_index();

function reset_start_index() {
    display_start_index = 0;
    display_end_index = display_start_index + filter_settings["hard_display_limit"] - 1;
}

if (filter_settings["timestamp_start"]){
    filter_settings["timestamp_start"] *= 1000; // in ms
};

$.get(address_book_path, function(respons) {
    address_book = respons;
    filter_metadata();
});

var all_folders = {{ all_folders|safe }};
```

```
for (id in all_folders) {
    $("#personal_folders").append('<a href="?folder='+ all_folders[id] +'">'+ all_folders[id].replace("_", "/") +'</a><br>');
}

function read_url(){
    var url = document.location.search;
    url = url.split("?")[1];
    variables = {};
    if (url != undefined) {
        parts = url.split("&");
        for (part_id in parts) {
            var part = parts[part_id].split("=");
            variables[decodeURIComponent(part[0])] = decodeURIComponent(part[1]);
        }
    }
    return variables
}
var http_get = read_url();


function filter_metadata() {
    filtered_messages = []; // reset
    for (id in all_messages) {
        message = all_messages[id];

        //folder filter
        if (http_get["folder"] != undefined) {
            if (message[EMAIL_FOLDERS_ID].indexOf(http_get["folder"]) == -1) {
                continue;
            }
        }

        //first recipient
        if (http_get["filter_address"] != undefined) {
            if (message[EMAIL_OTHER_PARTY_ID].indexOf(http_get["filter_address"]) == -1) {
                continue;
            }
        }

        //time filter
        if (filter_settings["timestamp_filter"]){
            var message_time = message[filter_settings["timestamp_choice"]] * 1000;
            var timestamp_start = filter_settings["timestamp_start"];
            if (timestamp_start) {
                // add next and previous week buttons
                if ( (message_time < timestamp_start) || (message_time > (timestamp_start + filter_settings["timestamp_duration"] *
                      86400000))) {
                    continue;
                };
            };
        };
        // only sent
        if (filter_settings["onlySent"]) {
            if (!message[DIRECTION_ID]) { continue; }
        };
        // only received
        if (filter_settings["onlyReceived"]) {
            if (message[DIRECTION_ID]) { continue; }
        };
        //draft
        if (filter_settings["isDraft"]) {
            if (!message[DRAFT_ID]) { continue; }
        };
        //mismatch
        if (filter_settings["isMismatch"]) {
            if (!message[EMAIL_MISSING_ID]) { continue; }
        };
        //attachment
        if (filter_settings["hasAttachments"]) {
            if (message[ATTACHMENT_ID] < 1) { continue; }
        };
        // direct message
        if (filter_settings["onlyDM"]) {
            if (message[NUM_INTERNAL_ID] + message[NUM_EXTERNAL_ID] != 1) { continue; }
        };
        // only added messages
        if (filter_settings["msg_added_filter"]) {
            if (!message[MESSAGE_ADDED_ID]) { continue; }
        };
        // must have internal receivers
        if (filter_settings["haveInternal"]) {
            if (message[NUM_INTERNAL_ID] < 1) { continue; }
        };
        // must have external receivers
        if (filter_settings["haveExternal"]) {
            if (message[NUM_EXTERNAL_ID] < 1) { continue; }
        };
        // forwarded
        if (filter_settings["onlyFW"]) {
            if (!message[FORWARD_ID]) { continue; }
        };
        // replied
        if (filter_settings["onlyRE"]) {
            if (!message[REPLY_ID]) { continue; }
        };
        //not fw or reply
        if (filter_settings["notREFW"]) {
```

```
            if (message[REPLY_ID] || message[FORWARD_ID]) { continue; }
        };
        // mass email
        if (filter_settings["mass_email_filter"]) {
            var num_rec = message[NUM_INTERNAL_ID] + message[NUM_EXTERNAL_ID];
            if (num_rec < filter_settings["mass_email_filter_limit"]) { continue; }
        };
        //only among suspicious
        if (filter_settings["suspects_only_filter"]) {
            if (message[TO_FROM_SUSPTECT_ID] == false) { continue; }
        };
        //word in subject
        if (filter_settings["word_filter"]){
            var subject = message[EMAIL_SUBJECT_ID];
            if (!(word_keyword in subject)){ continue; }
        };

        filtered_messages.push(message);
    };
    display_messages();
};


function display_messages() {
    display_end_index = display_start_index + filter_settings["hard_display_limit"];
    var number_messages = filtered_messages.length;
    if (display_end_index >= number_messages) {
        display_end_index = number_messages;
    };
    update_message_status(number_messages);

    var table = $("#table tbody");
    table.empty();
    var previos_day_str = "";
    for (var id = display_start_index; id < display_end_index; id++) {

        var MAX_LENGTH = 10000; //shall be found programatically, used for message length
        var message = filtered_messages[id];
        var number_internal_rec = message[NUM_INTERNAL_ID];


        var row = $( '<tr style="text-align: center;" data-idref="' + id + '" data-username="' + message[EMAIL_USERNAME_ID] + '" data-id="'
            + message[CONTENT_ID] + '" data-timestamp="' + message[filter_settings["timestamp_choice"]] + '" data-subject="' + message[
           EMAIL_SUBJECT_ID] + '"></tr>' ); //link to raw message

        // %%% case timeline %%%
        var html_content = "";
        html_content += '<td><a class="casetimeline">Add</a></td>';

        // %%% date, day and time %%%
        var utc_offset = new Date(message[filter_settings["timestamp_choice"]] * 1000).getTimezoneOffset() * 60000;  // in minutes -->
            millisec
        var time = new Date((message[filter_settings["timestamp_choice"]] * 1000) + utc_offset);
        var timezone = message[TIMEZONE_ID];
        if (timezone < 0){
            var timezone_str = " (-" + Math.abs(timezone) + ")";
        } else {
            var timezone_str = " (+" + Math.abs(timezone) + ")";
        }
        html_content += "<td>" + dateFormat( time, "yyyy-mm-dd") + "</td>";
        html_content += "<td>" + dateFormat( time, "ddd") + "</td> ";
        html_content += "<td>" + dateFormat( time, "HH:MM") + timezone_str + "</td>";

        // %%% other party (first) %%%
        var from = message[EMAIL_OTHER_PARTY_ID];
        //var from_html = stringLimit( from.split("@")[0].replace(".", " ").replace("_", " ") , 40);
        var from_html = stringLimit(from, 35);
        if (message[NUM_OTHERS] > 0) {
            var from_html = from_html + " (+" + message[NUM_OTHERS] + ")";
        }
        if (address_book[from]) {
            var add_timestamp = "/" + message[filter_settings["timestamp_choice"]] + "/";
            from_html = "<a href=\"/messages/" + address_book[from] + add_timestamp + "\" target=\"_blank\">" + from_html + "</a>";
        };
        html_content += "<td style=\"text-align: left;\">" + from_html + "</td>";

        // %%% internal receivers %%%
        html_content += "<td class=\"message_internal\">" + message[NUM_INTERNAL_ID] + "</td>";

        // %%% external receivers %%%
        html_content += "<td class=\"message_external\">" + message[NUM_EXTERNAL_ID] + "</td>";

        // %%% direction %%%
        var sent = message[DIRECTION_ID];
            if (sent) {
                html_content += "<td><img src=\"/static/inbox.png\"></td>";
            }
            else {
                html_content += "<td><img src=\"/static/sent.png\"></td>";
            }

        // %%% subject %%%
        var subject = stringLimit(message[EMAIL_SUBJECT_ID], 55)
        html_content += '<td style="text-align: left;">' + subject + '</td>';

        // %%% num_attachments %%%
        var attachments = message[ATTACHMENT_ID];
```

```
if (attachments != null) {
    var num_attachments = attachments.length;
} else {
    var num_attachments = 0;
}
html_content += '<td class="message_attachments">' + num_attachments + '</td>';

// %%% forward %%%
if (message[FORWARD_ID]) {
    forward_html = check_image;
} else {
    forward_html = "";
}
html_content += "<td>" + forward_html + "</td>";

// %%% reply %%%
if (message[REPLY_ID]) {
    reply_html = check_image;
} else {
    reply_html = "";
}
html_content += "<td>" + reply_html + "</td>";

// %%% payload length %%%
html_content += "<td class=\"message_length\">" + message[PAYLOAD_LEN_ID] + "</td>";

// %%% suspect flag %%%
var other_party_suspect = message[TO_FROM_SUSPTECT_ID];
if (other_party_suspect){
    other_party_suspect = '<img src="/static/user.png">';
} else {
    other_party_suspect = "";
}
html_content += "<td>" + other_party_suspect + "</td>";

// %%% added back — deleted %%%
var added_later = message[MESSAGE_ADDED_ID];
if (added_later) {
    html_content += "<td><img src=\"/static/trash.png\"></td>";
} else {
    html_content += "<td></td>";
};

// %%% draft %%%
var draft = message[DRAFT_ID];
if (draft) {
    draft = "<img src=\"/static/draft.png\">";
} else {
    draft = "";
}
html_content += "<td>" + draft + "</td>";

// %%% mismatch of email address
var mismatch = message[EMAIL_MISSING_ID];
if (mismatch) {
    mismatch = "<img src=\"/static/mismatch.png\">";
} else {
    mismatch = "";
}
html_content += "<td>" + mismatch + "</td>";



// %%% messages per day %%%
html_content += "<td class=\"message_perday\">" + message[MESSAGES_DAY_ID] + "</td>";

// %%% messages per week %%%
html_content += "<td class=\"message_perweek\">" + message[MESSAGES_WEEK_ID] + "</td>";

// %%% people per week %%%
html_content += "<td class=\"message_peersweek\">" + message[UNIQUE_PPL_ID] + "</td>";



// %%% in folder %%%
var folders = message[EMAIL_FOLDERS_ID];
var folder_html = "";
for (folder_id in folders){
    folder_html += '<a href="?folder=' + folders[folder_id] + '" style="margin-right: 10px;">' + folders[folder_id].replace("_", "/
        ") + '</a>';
}
//folder = folder[folder.length - 1].replace("_", " ");




if (dateFormat(time, "yyyy-mm-dd") != previos_day_str) {
    row.css("border-top", "1px solid #222");
}
previos_day_str = dateFormat(time, "yyyy-mm-dd");

var check_image = '<img src="/static/check.png">';

html_content += "<td>"+folder_html+"</td>";
```

```
        function include(arr,obj) {
            return (arr.indexOf(obj) != -1);
        }

        color_code_row(row, message[CONTENT_ID]);

        row.html(html_content);
        table.append(row);
    }


colorcode(".message_length");
colorcode(".message_attachments");
colorcode(".message_internal");
colorcode(".message_external");
colorcode(".message_perday");
colorcode(".message_perweek");
colorcode(".message_peersweek");
colorcode(".message_attachments");

hookRawDisplay();
hookAddCaseTimeline();
}

function hookAddCaseTimeline(){
    $("#table td .casetimeline").click(
        function (){
            var this_row = $(this).parent().parent();  // need to go back two steps to reach the row
            //var userid = this_row.data("username"); this was a bad idea
            var file_id = this_row.data("id");
            var comment = this_row.data("subject");
            var timestamp = this_row.data("timestamp");
            console.log(user_id)

            $.ajax({
                url: "/add_casetimeline/",
                type: "POST",
                data: {
                    userid: user_id,
                    fileid: file_id,
                    timestamp: timestamp,
                    comment: comment,
                },
                success: function(response) {
                    if (response == "success") {
                        timeline_messages.push(file_id);
                        console.log(timeline_messages)
                        color_code_row(this_row, file_id);
                    }
                    console.log(response);
                }
            });
        }
    );
}

function hookRawDisplay(){
    $("#table tbody tr").click(
        function () {
            var this_row = $(this);
            var idref = this_row.data("idref");
            var userid = this_row.data("username");
            var file_id = this_row.data("id");
            var full_path = "/static/content/"+ userid +"/"+ file_id +".json";


            var notify_url = "/mark_message/"+ user_id +"/"+ file_id +"/";


            $.ajax({
                url: notify_url,
                type: "GET",
                data: "",
                success: function(response) {
                    console.log(response);
                }
            });


            var attachments = filtered_messages[idref][ATTACHMENT_ID];
            if (attachments.length > 0) {
                var attachment_html = "";
                for (var item in attachments) {
                    var attachment = attachments[item];
                    name = attachment[0];
                    md5 = attachment[1];
                    application_type = attachment[2];
                    attachment_html += '<a href="/static/attachments/'+md5+'">'+name+'</a><br>';
                }
            } else {
                attachment_html = "No attachments found."
            }
            $('#attachment_box').html(attachment_html);
            $('#header_box_details').html('<a href="'+ full_path +'">Direct link</a>');

            $.ajax({
                url: full_path,
```

```
                type: 'GET',
                data: '',
                success: function(response) {
                    console.log(typeof(read_messages))
                    read_messages.push(file_id); //mark message as read only if it loads

                    var header = response["header"];
                    var text_plain = response["plain"];
                    var text_html = response["html"];

                    if (text_html == null) {
                        text_html = "<html><body><p>This message has no HTML parts</p></body></html>";
                    }
                    if (text_plain != null) {
                        text_plain = text_plain.replace(/\n/g, '<br>');
                    } else {
                        text_plain = "This message has no plain-text part";
                    }

                    var header_str = ""
                    for (var key in header) {
                        value = header[key];
                        value = value.replace(">", "&gt;").replace("<", "&lt;");
                        if (key == "To" || key == "From" || key == "Cc" || key == "Bcc") {
                            var email_replace = "";
                            var receivers = value.replace(/ /g,'').replace(/(\r\n|\n|\r)/gm,"").split(","); // remove spaces, new lines and
                                    split individual addresses
                            for (var email_key in receivers) {
                                var email_address = receivers[email_key];
                                email_replace += emailToLink(email_address) + ", ";
                            }
                            value = email_replace;
                        }
                        header_str += "<b>" + key + "</b>: " + value + "<br>";
                    }


                    $('#header_box').html(header_str);
                    $('#plain_box').html(text_plain);

                    // http://www.zomeoff.com/javascript-how-to-load-dynamic-contents-html-string-json-to-iframe/
                    //var iframe = document.getElementById('html_box_iframe');
                    var iframe = $('iframe#html_box_iframe').get(0);
                    var iframedoc = iframe.document;
                    if (iframe.contentDocument)
                        iframedoc = iframe.contentDocument;
                    else if (iframe.contentWindow)
                        iframedoc = iframe.contentWindow.document;
                    if (iframedoc){
                         iframedoc.open();
                         iframedoc.writeln(text_html);
                         iframedoc.close();
                     }


                    if (lastViewed != null) {
                        color_code_row(lastViewed, lastViewed.data("id"));
                    }
                    $(this_row).css({
                        "background": color_active,
                    });
                    lastViewed = $(this_row);
                },
                error: function(e) {
                    console.log(e.message);
                }
            });
        }
    );
}

function update_message_status(total) {
    if (total > 0) {
        if (display_end_index > total) {
            display_end_index = total;
        }
        start = display_start_index + 1; //0-indexed
        end = display_end_index;

        var previous = "back a page";
        if (start > 1) {
            previous = "<a id=\"previous_page\">back a page</a>";
        };
        var next = "next page";
        if (end < total) {
            next = "<a id=\"next_page\">next page</a>";
        };
        var pagination_string = "Showing message " + start + " to " + end + " of " + total + ". (" + previous + " | " + next + ")";
    } else {
        pagination_string = "No messages to display";
    }

    $("#pagination").html(pagination_string);

    if (filter_settings["timestamp_filter"]){
        timestamp_start = filter_settings["timestamp_start"];
        timestamp_duration = filter_settings["timestamp_duration"];
```

```
        var previous_period = (timestamp_start - (timestamp_duration * 86400000)) / 1000; // sec per day
        var next_period = (timestamp_start + (timestamp_duration * 86400000)) / 1000;
        var previos_link = '<a href="/messages/'+ user_id +'/'+ previous_period +'/">Earlier</a>';
        var next_link = '<a href="/messages/'+ user_id +'/'+ next_period +'/">Later</a>';
        $("#pagination").append("<br>Time filter: " + previos_link + " | " + next_link);
    }

    $("#next_page").click(function(){ next_page(); });
    $("#previous_page").click(function(){ previous_page(); });
}

function next_page() {
    display_start_index += filter_settings["hard_display_limit"];
    display_messages();
}

function previous_page() {
    display_start_index -= filter_settings["hard_display_limit"];
    if (display_start_index < 0) {
        display_start_index = 0;
    }
    display_messages();
}

// helper functions
function emailToLink(email) {
    if (address_book[email]) {
        var username = address_book[email];
        var path = "/messages/" + username + "/";
        return "<a href=\"" + path + "\" target=\"_blank\">" + username + "</a> (" + email + ")";
    }
    else {
        return email;
    }
}

function stringLimit(string, limit) {
    if (string.length > limit) {
        return string.substring(0, limit - 1) + "..";
    }
    else {
        return string;
    }
}

//duplicate code
function colorcode(class_label) {
    var maximum = 0;
    var elements = $(class_label);
    elements.each(function(){
        var value = parseInt($(this).html());
        if (value > maximum) {
            maximum = value;
        }
    });

    elements.each(function(){
        var pct = ($(this).html() / maximum).toFixed(3);
        $(this).css("background-color", "rgba(255,0,0," + pct + ")");
    });
}

</script>
{% endblock %}
```

E.25 server/templates/casetimeline.html

```
{% extends "index.html" %}

{% block html_css %}

    <link rel="stylesheet" type="text/css" href="/static/timeline.css">
    <style>

        #mytimeline{
            width: 100%;
            height: 100%;
        }

    </style>

{% endblock %}

{% block html_body %}

    <div id="mytimeline"></div>

{% endblock %}

{% block html_javascript %}
<script type="text/javascript" src="/static/timeline.js"></script>
<script type="text/javascript">

$( document ).ready(function(){
```

```
    var timeline;
    var data;
    drawVisualization()

    function drawVisualization() {
        data = [
            {% for message in data %}
                {
                'start': new Date({{message.timestamp}} * 1000),
                'content': 'E-mail message:<br>Subject: <b>{{ message.comment|escapejs }}</b><br><a target="_blank" href="/messages/{{
                       message.user_id }}/{{  message.timestamp }}/">Message link</a> | <a target="_blank" href="/static/content/{{
                       message.user_id }}/{{  message.file_id }}.json">Direct link<a/>'
                },
            {% endfor %}
        ];
        var options = {
            'editable': false,   // enable dragging and editing events
        };
        timeline = new links.Timeline(document.getElementById('mytimeline'));
        function onRangeChanged(properties) {
            document.getElementById('info').innerHTML += 'rangechanged ' +
                    properties.start + ' - ' + properties.end + '<br>';
        }
        links.events.addListener(timeline, 'rangechanged', onRangeChanged);
        timeline.draw(data, options);
    }
});
</script>
{% endblock %}
```