Understanding Internet protocol design decisions

Cezary Radoslaw Jaskula
University of Oslo
Department of Informatics

2020-12-01

Chapter 1

Introduction

The high-level goal of this project is to develop methods and tools that will allow us to make out the arguments and decision making contained within the Internet Engineering Task email archives. By doing this we hope to gain a better understanding of why both modern and old internet protocols function the way they do.

1.1 The IETF

IETF stands for Internet Engineering Task Force, and is an organization composed of network designers, vendors, scientists and operators concerned with evolving the internet architecture, as well as ensuring smooth operation of the internet. The participants are organized in "Working groups", these groups are organized based on research topics in specific areas. The work is mainly being done by the use of mailing lists, as well as organized meetings 3 times a year. These working groups are grouped into areas, and managed by area directors, or "AD"-s. ADs are members of the "Internet engineering steering group" (IESG).

1.2 RFCs

RFC stands for "Request for comments" and is a type of publication used by the IETF containing documents, comments and discussions concerning internet protocols, procedures, programs, concepts and so on. The IETF has put in place a search engine for easy access to these publications which can be found at

RFC Search

Alternatively, the entire RFC stream can be found at the following link:

rfc-editor.org

An RFC always begins its life as an “internet draft”, these can be submitted by anyone and there are 2 ways of doing so. The first one is to use the IETFs I-D Submission Tool, or IDST for short, to submit the internet draft. The second is to simply send the Internet draft the following email address:

internet-drafts@ietf.org

The IETF however recommends using the first method mentioned as manually validating an internet draft sent in by mail takes significantly more time than processing it automatically. The drafts are however not reviewed in any way, as long as they fulfill the format rules.

1.3 The lifecycle of an internet draft

When an internet draft is submitted, it is first checked to see if it fulfills the format rules, and is accepted if this is does. At this point the internet draft will get a title that follows the following format:

Draft-lastname-intendedworkinggroupname-nameofidea-00

The last two numbers represent the revision number, meaning that if a new internet draft is submitted and goes as far as being revised, its new title will end in 01. All internet draft are therefore considered to be a work in progress, and must be cited as so. Citing an internet draft is not always a good idea however, as all internet draft older than 185 days are removed. This is not guaranteed to happen, we can still find internet drafts from over 20 years ago with a simple google search, there is however no guarantee that any specific internet draft will survive.

For an internet draft to become an RFC, a working group has to decide to work on it, if this happens the internet drafts title is again changed, this time the indendedworkinggroupname is changed to the “ietf-” and the actual name of the group that will be working with this internet draft, and it replaces the author’s name, meaning in now looks like this:

Draft-ietf-workinggroupname-nameofidea-00

The last two digits form a number that is incremented as described previously, but the counter is reset to zero whenever the name changes (i.e., also at working group acceptance). Once a working group has decided to focus on a specific internet draft it is highly likely that it will end up as an RFC. RFCs in contrast to internet drafts do not change nor expire, meaning that we can easily find any RFC ever published by the IETF on their website.

Since RFCs are the final version of internet drafts, figuring out a way to access these RFCs will be crucial as they contain huge amounts of data concerning things going on in the IETF. One way of achieving this would be to download the entire archive and store it locally, then process the data and insert it into a searchable database system that our machine learning algorithm can use to accomplish its task.

Chapter 2

Groups within the IETF

2.1 Internet engineering steering group

As the name implies this group is in charge of steering the IETF, mainly focusing on technical management and the internet standards process. The IESG consists of the Area Directors (ADs) who are selected by the Nominations Committee (NomCom) and are appointed for two years.

2.2 Internet research task force

The internet research task force is a separate but at the same time parallel organization to the IETF that focuses on long term research topics.This organization is concerned with researching internet protocols, applications, architecture and technology. The members of this organization are divided into research groups, where they get the long term membership needed for research and collaboration. As with the IETF in general, participation in this group is done only by individuals, and not groups or organizations.

The difference between this organization and the IETF is that the IRTF is concerned with long term topics, whereas the IETF is more focused on short term problems, as well as developing standards.

2.3 Tools team

The tools team consists of volunteers from the IETF that develop tools needed by the IETF. The team leader, as well as the team members are selected by the General Area Director. The team investigates open source tools that can help other members of IETF to work more efficiently. When no suitable tools are found, the team develops and maintains custom tools, standardized tool are however greatly preferred.

Some of the tools team projects are:
Dataracker
The IETF Datatracker is the biggest custom tool developed and maintained by the Tools Team. It is the most visible and heavily-used tool of the IETF. The Datatracker is used to upload Internet-Drafts, manage their review and approval, manage meeting materials, and manage working groups. Feedback on the Datatracker is actively solicited, and the Tools Team is constantly working on bug fixes and enhancements.

Postconfirm
The Postconfirm system employs a variety of verification methods to discard unwanted email.

Mail archive tool
The Mail Archive tool provides advanced, easy-to-use archive searches for all IETF lists, past and present. This tool is under active development, with many new features planned.

Mailman
Mailman is a mail manager used by the IETF to archive communication done by email, as well as manage the various mailing lists.

Xml2rfc
A tool widely used by the IETF community to write internet drafts

RFCdiff
The RFCdiff tool allows easy comparison of the changes between two versions of plain text documents.

Source : https://www.ietf.org/about/groups/tools/

2.4 Internet Architecture Board

The Internet Architecture Board is a committee of the IESG, its responsibilities are :
IESG confirmation : Confirming the IETF chair directors as well as IESG Area directors
Architectural oversight : Providing oversight and comments on spects of the architecture of protocols and procedures used by the internet
Standards process oversight and appeal : Providing oversight over the process of creating internet standards, as well as acting as an appeal board for complaints of improper execution of said standards

RFC : IAB is responsible for editorian management and publishing of RFCs

External Liaison : the IAB is responsible for representing the IETF's interests in liaison relationships with other organizations concerned with issues relevant to the world-wide-web.

Advice to ISOC : Acting as a source of guidance to the Board of Trustees and Officers of the Internet Society concerning procedural, technical, architectural and policy matters.

2.5 Directorates

Directorates are comprised of experienced members of the IETF and often serve as advisors for IETF work. A directorate is defined by the IETF as follows:

"In many areas, the Area Directors have formed an advisory group or directorate. These comprise experienced members of the IETF and the technical community represented by the area. The specific name and the details of the role for each group differ from area to area, but the primary intent is that these groups assist the Area Director(s), e.g., with the review of specifications produced in the area."

Source: https://www.ietf.org/about/groups/directorates/

Chapter 3

A full-text database

To make the IETF's mail archive usable we will have to feed it into a searchable full text database of our choice, namely "Solr". Solr is an open source search platform written in Java, from Apaches Lucene project. Its features that are most noteworthy for this project are full-text search, real-time indexing and dynamic clustering. These will allow to easily search the archive and extract useful information such as word count and frequency, amount of emails sent from a certain mailing list and many other statistics that are sure to prove useful in the future.

3.1 The email structure

As previously mentioned the IETF uses their own email program (called mailman) to handle their communications, this is both an advantage and a disadvantage.

It makes it easier to extract the important parts of the email, such as the ID, date of sending, title, subject line and so on. To our disadvantage, a lot of conversations do not start with an email sent from the mailman program, but rather a different service, where after a while the conversation is redirected to mailman. This leads to emails in a lot of different formants due to various email clients used, as well as the various ways these email client can be used.This however could be used to track down information, if for example a person is very active in a conversation related to a very specific topic, we can simply search for the specific format that person is using to possibly find more information related to that topic. This could prove to be useful if a person is using several different email accounts or their emails are missing signatures. A lot of users also include their signature in their emails, often containing their full name, phone number and if relevant, their position and the company they are affiliated with. With this information we can accurately track which person engages in which discussion, and what topics are relevant for this person, further expanding the amount of data we can obtain from these emails.

Let’s take a look at how exactly an email sent from the mailman program looks:

The first few lines are as follows:

From nobody Wed Mar 14 02:56:33 2018
Return-Path:
X-Original-To:
Delivered-To:
Received:
X-Virus-Scanned:
X-Spam-Flag:
X-Spam-Score:
X-Spam-Level:
X-Spam-Status:
Received:

These do not carry and significant information and can be simply ignored for the time being. The next fields are as follows:

To:
From:
Message-ID
Date:

These are rather self explanatory, one thing that should be noted is that the date field includes information about the time zone the email was sent from.

Next, there are:

User-Agent:
MIME-Version:
X-Provags-ID:
X-UI-Out-Filterresults:
Archived-At:
Subject:
X-BeenThere:
X-Mailman-Version:
Precedence:

Aside from the subject field there seems to be no relevant data in the other fields.

Lastly, there are a few fields related to the list itself:

List-Id:
List-Unsubscribe:
List-Archive:
List-Post:
List-Help:
List-Subscribe:
X-List-Received-Date:

Here only List-id and X-List-received-Date seem to have any relevance.

The next thing that needs to be resolved before we can start dissecting the emails and feeding them into our database is attachments. Attachments are stored in the archived emails as their ascii representations, meaning that if an attachment is present in a conversation, it creates a lot of visual garbage. One possibility would be to find a library that would allow us to convert these files back to their original state, and then analyze their contents, this however is a whole different task altogether as it would again require to parse many different files in many different formats. For all these reasons attachments will be ignored while parsing.

Chapter 4

Roadmap

4.1 Step 1

To even attempt to organize the data we first need to store it somewhere where it can be easily accessed and cannot be altered by unauthorized entities. To accomplish this the “wget” linux command will be put to use as it will recursively download the entire archive onto a storage medium of our choice. At start this will be nothing more than a flash drive or a small portable hdd/ssd, it will however be uploaded to ifi’s servers at a later date. The specific wget command is as follows :

wget -r https://ietf.org/mail-archive/text/

4.2 Step 2

Once the archive is stored in a safe place the next step is to extract the parts that are useful for our research. The entire archive will have to be parsed in order to obtain said parts and feed them into our Solr database. To accomplish this a parser will be written in the Python programming language,this program will utilize Pysolr package to feed the data into Solr. What is considered to be useful is as follows :

To:
From:
Message-ID
Date:
The message text itself Any signatures the email may contain If the email contains a conversation that was started outside of the mailman program

As previously mentioned email attachments may prove to be useful later but are of no concern at the current time as restoring and analyzing them will require amounts of time and work that are currently unjustifiable.

4.3 Step 3

At this point the preparation are done and we can begin looking into how to retrieve the desired data.This can be done in vast amount of different ways that will all need to be considered, this brings us nicely to the next part.

Chapter 5

The toolbox

5.1 Similarity

Similarity can between words and sentences can be measured in a lot of different ways, and is something our search tool will need the ability to do in order to provide the best possible result. We may be interested in email that contains a string similar to our search string for example, or we may want to determine which emails contain similar words and sentences.

First method of calculation that comes to mind is calculating the Jaccard similarity coefficient. This method works by first lemmetizating the two sentences we want to calculate, then determining the amount of overlap in these two sentences. Cosine similarity could also be used, this method differs mainly from the previous one in that is compares the similarity of vectors, meaning that if we are to compare sentences, then would first have to be made into vector. These vectors however need to contain numbers, not words,that means the using term frequency, or inverse document frequency can be used to fill the vectors. This method is more suited for large collections of text rather than single sentences, this should however not pose a problem in our case as we can get that information from Solr.

5.2 Topic modelling

As the name suggests this is a method of determining the topic of a document. This method works of the intuitive notion that given a document on a certain topic, the document will contain more words related to that topic than documents that relate to other topics. For example, a document about plants will have a more frequent use of words such as “plant” , “soil” and “seeds” than a document about making ice cream. Specific words are however not limited to only one specific topic , therefore it may seem as if a document is discussing several different topics, this is solved by simply calculating the probability based on the word frequency and picking the most likely outcome. I suspect topic modelling will be a very useful tool in organizing the mail archives.

5.3 Argumentation mining

Argumentation mining is a research area within the field of natural language processing.The goal of argumentation is to identify and extract argumentative structures. This is very relevant to this thesis as being able to accurately detect where discussions take place, and extracting the arguments used in that discussion will allow us to determine why certain decisions were made, or possibly more importantly, why other suggestions/proposals were discarded..

5.4 The role of ML in this project

To better understand the role that Machine Learning (ML) can take for this work, we first need to look at what kinds of problems machine learning can help us solve. The most common one is classification, it is widely used in image recognition, search engines and so on.

Can our task be reduced to a classification problem ?.

Classifying each mailing list as their own class does not provide any useful results, as that task has already been done for us by the IETF. A given mailing list does not always stick to one topic however. This is where we could use ML to figure out what exactly is being discussed in the email, and if its relevant to the topic the mailing list was originally intended for.. There is also the possibility of several other mailing lists discussing the same topic, possibly approaching it from a different angle, meaning that they have the possibility of containing valuable insight to the specific topic. This also means that the relevant emails from all of the relevant mailings lists should be returned when looking for a specific topic. This is where topic modelling could be applied, as it is a machine learning algorithm

Chapter 6

Machine learning, how to apply it

SUPERVISED LEARNING
The main problem with this method is that it requires large amount of labeled data for both training, testing, and validation of results, something we in this case do not have access to. Making a program that labels the data could be done, but then, how to we know what to label the data?

UNSUPERVISED LEARNING
This method is much more suitable for our purpose as it does not require anything more than unlabeled data, something we have plenty of.

SEMI-SUPERVISED LEARNING
For the time being i see no use of combining supervised and unsupervised learning, since we will always run into the problem of unlabeled data, and it that problem is somehow solved we could just apply regular supervised learning instead.

One thing that could potentially be useful and/or helpful is some way would be using an unsupervised machine learning algorithms like K-nearest neighbour to classify the emails, then try to label them based on their contents, then having labeled the data, train a neural network based on those labels.

REINFORCEMENT LEARNING
Reinforcements learning works by maximizing some notion of a reward by taking action in the environments the algorithm is confined to. To accomplish this there first needs to exist a way of calculating the potential reward given an action, and this is where our problems begin. Reinforcement learning is mostly suited to optimization problem, such as playing games where one can mathematically calculate the next best move, congestion control and similar cases where determining the next action is relatively easy. In our case measuring the quality of our outcome is the main challenge.

So, how do we measure the quality of the outcome ? Since (at least at first) we will be working with a full-text searchable database, the first step in determining the quality of the databases returned emails, but to have any emails returned we first need a query. This means that what we are working towards is learning how to write good and accurate queries, queries that have high probability of returning the information we need.

But still, we will evaluate the quality of the query based on the quality of the dataset returned by the database, by executing said query, so the original problem remains.

We could use thread detection to see if the returned dataset contains a discussion, as a discussion often takes place before something is decided upon. The problem here is determining what exactly was decided, since there is no guarantee that what is being discussed is actually related to what was searched for. This means that we would have to look for a thread in the dataset, then, determine what is being discussed, and if it is related to the query, as well as how strongly related it is. Calculating similarity between the query words and the words found in the thread could be calculated as mentioned before. But again, as not all words carry the same weight, and neither do word similar to them. We could classify words based on topics they are highly related to, then focus specifically on them, but that is a whole different problem in of itself. Even if we manage to overcome these challenges there is still the question of how to pick out the one email where the decision took place, and if there is even any point in doing so, as an email containing "Yes, do it that way" isn't very helpful. Returning the whole discussion seems like a better choice here, since the goal of this project it to gain an understanding of why certain protocols are the way they are, getting to know what ideas were suggested, and why they were scrapped or not, is very valuable, this however can only be obtained be looking into the entire discussion, not just one single email. For this reason I believe that the best solution here is looking for the beginning and end of a discussion, instead of a single email.

One of the big challenges with this idea is that there are a lot of things being discussed at the same time, in the same mailing lists.

Chapter 7

chapter 3

Chapter 8

Preparations

In order to start working with the text we first need to have an easy and most importalty , quick way of accesing the data as even small delays will add up fast concidering the sheer ammount of text we are trying to process. To accomplish this it is preferable to have the entire archive stored locally on our own machine, or, even better, a fast server with nvme storage to further descrease execution time. I will also be doing utilizing a version of red hat linux in this thesis.

The next step is to actually get the mail archive saved ou our local hardware, whatever that may be, since we are using linux, we will utilize the tools it provides to help us along the way. The first one we will need is the "wget" command. From `https://www.gnu.org/software/wget/manual/wget.html#Overview` GNU Wget is a free utility for non-interactive download of files from the Web. It supports HTTP, HTTPS, and FTP protocols, as well as retrieval through HTTP proxies.

8.1 Wget flags

Wget as most linux commands has a set of "flags" or options we can set to modify how the commands will behave, it this case however we are only interrested in the "-r" flag. -r means "recursive" unless this option is sett, the wget command will only download the exact site it is pointed to and nothing else. With the -r flag set however, the command will recursively download everything it can get from a given starting point. This however can be a problem in it self, how to we avoid goin in circles for example ? Luckily for us, this is not a problem we need to concern ourselfs with, as the arvhives to not contain link leading to websites other then the archive itself, and any links contained within the m-boc files themselv are irrelevant, wget does not inspect the files, it only downloads them.

Finally we can execute the following command

wget -r https://ietf.org/mail-archive/text/

As of the day i am downloading this archive there are 77224 files in this archive, this means 77224 mailboxes, that all need to be parsed, and inserted into solr to make them searchable

8.2 Solr

As previously mentioned Solr is the program we will be using to manage all of our data. However, do be able to use Solr efefcitvely, we first need to familiarize ourselfs with it. The first thing we need to understand is that Solr has nothing in common with SQL or relational databases, it is purely focused on fast and effitient text search. This means that there is no need to define relations between the columns in the databse, there is no need to define a Primary and seconadry key, in fact the closest thing Solr has to a priamry key, is the unique document id that is automatically generated and assigned to each document.

To start working with solt we first need to install it, the program can be downloaded free of charge from the following link `https://archive.apache.org/dist/lucene/solr/`

For this thesis i wiill be using Solr versio 8.2.0 For Solr to be able to run however, we will first need to install the latest version of java as well, the program is written in java after all, and will not be able to run without java beeing present on opur machiene. Java can be downloaded from the following url `https://www.java.com/en/download/`

Once Java is downloaded and installed we can go back to Solr, to start Solr we need to go to the bin folder inside the Solr directory, and use the following command

solr start

After about 1 minute the program wil be up and running, to access the graphical interface that Solr provides a internet browser is needed, any modern browser should be good enough,for consistency sake i will be using firefox in this thesis. In the browsers adress field type the folowing :

localhost:8983

localhost beeing the address of the machiene we are currently using, and 8983 beeing the deafult port used by solr, it is also the port i will be using in this thesis.

Solr uses something called cores, cores can be seen as iondividial databases managed by the program, they are independant of eachother, and searches cannot be done across several cores at the same time. To create a new core we need to navigate to the "Core Admin" tab, once there, click on "Add Core", and fill out the cores name and directory, but dont create the core yet. Before the core can be created we need to actualyl create the directory itself, the root needs to have the same name as the core we want to create. ASdditionaly, a core has its own schema, and config file, i will go into further detail as to how these work when it becomes relevant, for noe we will stick to the deafult setting provided by solr. To use these setting we must navigate to their directory:

`solr-8.2.0\server\solr\configsets\_default\conf`

Copy the entire conf folder and paste it inside the core folder, once this is done we can finalyl go back to the web interface and cllick "Add Core"

One created the core can be seen in the web interface by selecting in from the drop down menu on the left side. From there we can modify the core, see statistics, and send queries.

Now that we have solr up an running, the nxt step would be preparing data to feed into it

Chapter 9

Parsing

Before we can start parsing the mbox files, we first need to establish what kind of information we are interrsetyed in extracting, and why. I will be using RFC 4021, which defines the header fields, it should be noted that this rfc has been updated by rfc 5322,and updated again by rfc 6854, these updatse do not affect the meaning og data contained in the header fields however.

First off all, we need a way to track the time at which a mail was sent, this information is contained in the "Date:" header, next we need to know who sent it, and who the intended recipent is, as we can use this information to bind individals to specific mailing lists, we can use this inforamtion to keep track of who is contrubiting to what and so on. We can extract this information from the "From:" field.

Another way we could group individuals is by checking if the same agent has been respnsible for trasmitting their messages, this might be the case if a group if individuals all for for the same organization for example. Luckily for is this information is contained within the "Sender:" field. It is important not to confuse the "From" field with the "Sender" field, as they might seem to be describing the same thing, the difference between in that "From" refers to a specific user, the author of that message. The "Sender" field however refers to the agent responsible for trasmitting the message, meaning the two field will in many cases be the same, but there is no guarantee that they will.

In order to be able to keep track of a conversation happening by mail we need to know who is replying to who, the "Reply-to" field is exactly that. But cant we just use the mailinglist address to keep track fo the conversation ?, the simply answer is no. While this is certainly valuable information, there are a lot of cases where a conversation takes place outside of the mailing list, and is at some point redirected to the mailing list, possibly to again be continued outside, and redirected again. There is still a lot of information we can extract to help us keep track of covnersations, namely the "To:","In-Reply-To:", "Cc:" and "Subject", "To:" specifically tell who the message is intended for, "Cc:" spesifies who this message may concern, and who is a part of the conversations, "Subject:" specifies the topic of the conversation. The "In-Reply-To:" field specifies a mailbox for replies to massage, this will in our case often be the mailing list.

Lastly we need to know which mailing list a message belongs too, this can be aquired both from the "In-Reply-to:" field, as well as the download directory, as all mbox-es belongig to a sepcific mailing list are located in a folder with the mailing lists name, all we need to do is some simple string editing, and since the "In-Reply-To:" isnt im my experience very reliable, i will be using both methods.

Each message alse has its own "Message-Id", this can proove to be useful for idetification at a later point, and will therefore be included in the database.

Last and defiently not least we need the actuall content of the message.

We will end up with the following fields `Date`
`From`
`Sender`
`Reply-to`
`To`
`Cc`
`Message-ID`
`In-Reply-To`
`Subject`
`Mailing-list`
`Content`

as well as

`id`
`_version_`

which are the fields automatically added by Solr additionally, a "file-directory" field will be included as it will be usefull during debugging

9.0.1 Parsing the content

The next challange is deciding how to parse the content field in the emails. The main problem is this case is how various messanges are formatted, some are pure text, some contain images, attachments, and some are MIME messanges.

From wikipedia:
"Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the format of email messages to support text in character sets other than ASCII, as well attachments of audio, video, images, and application programs. Message bodies may consist of multiple parts, and header information may be specified in non-ASCII character sets."

This means that not only do we need to somehow figure out what text to keep, but also what to do with the multimedia parts of a given message.We also need a way to keep track of all the parts of a mime message, and a way to correctly assemble them back to their original form ,if possible.

9.0.2 Attachemnts

The IETF mesasge archive saves all of the data in pure text, this means that any and all attachments, no matter the kind are still present in the archive, but only in their ascii representation. As one may imagine they also take up a lot of space,both concidering the actuall space on the storage mediam, as well as space in the document itself. What we then end up with is an mbox file that may seem to contain a lot of datn, only for 90 percent gibberish that can not be red by a human, thats on top of headers and a lot of other information that makes the raw mbox files hard to read. But how do we handle these attachemnts then ?, the simplest solution would be to simply ignore them, we will however be throwing away a lot of potentially very valuable information in this case. One could attempt to recreate the the files back to their original state, then anylize them, this however could be a another tehsis on its own, i therefore bleave it is outside the scope of this specific thesis. Dpending on the outcome and discoveries made here, it may be worth it to attempt this aproach however. The same argument can be used for the mulitmedia parts of the MIME messanges. So, for simplicitys sake, i will be ignoring all parts of the messages payload that were not originalyl in text form, that means, anything but the written message itself will be ignored.

Another common problem one will encounter when trying to extract the text is how to distuinguish, for example, HTML from regular speech ?, onw way would be to use machiene learning, train in using varous html exambles, then make it filter out the html sections. Doing this by hand would take too mutch time concidering the sheer ammount of text that has been generated by over 30 years of communication. This might however not be too big of a problem, as computer code is something that is most likely to appear in MIME messages, and there are standard libraries for parsing MIME messanges, using those should filter out most, if not all of the code.

There are a lot a things one could attempt to do while parsing that im not going to attempt at the moment, like hyperlink detection or the previously mentioned restoration of files. The primary goal is the extract as mutch as possible of the actual message text, and insert it ontop solar in a format that is easly readable by a human.

9.1 The parser

Before getting to the code we needan outline of how the program is going to function, and why, as well as concider what packages to use.

As we will be working with mailboxes, we need a way of extrating data from them, the standard python package "mailboxl" allows us to make an itterator based on a gicen mbox files, that we can use to itterate over the messanges one at a time. Once we have one specific email, we need a way of getting information out of it, the standart python package "email" can help with this, as it allows us to extract specific header fields, as well as handle MIME mail. I will also be usen the time package to measure the parsers running time, but this is not strickly needed for the program to do its job propperly. I will also assume that the the IETF archive downloaded is placed in the same directory as the program files.

9.1.1 parser outline

The general flow of the program is as follows:

1.locate all files with the extention ".mail", these are the mbox files we had downloaded earlier.
2.initialize a array that will contain the parsed messanges
3.for each file, do the following :
3.1 Extract the mailing lists name from the files path 3.2 create and itterator using the mailbox package and the mailbox file as argument
3.3 use the itterator to fetch a messange from the mailbox
3.4 extract the prvously described headers using the get() command
3.5 determine if the message is a plan text message, or a MIME message

3.6
if the message is plan text
use `get_payload`)) to get the contents

if the message is MIME
create an itterator that allows is to itterate over all parts on the message
using this itterrator, check if the content type is "text/plain"
if true, add the content of that part to an accumulator variable
to this untill all parts have beein investiagted

3.7 add the message to an accumulator variable
3.8 go back to ste 3.3 and reapet untill no new mesages can be found

4. upload the parsed messages to solr

9.1.2 UnicodeEncodeError

[language=Python] This is a very common error one will encounter while trying to parse the data, it is caused by the mailbox package, as the itterator will attempt to re-encode the each email. There is howeer a simple way to fix this, as python allows us to set the default action to perform when we encounter this error, this is usually set to, thorwing the error, bit it can be set to automatically replace the problematic character with a special character ment for this exact purpose.

```
codecs.register_error("strict", codecs.replace_errors)
```

"strict" meaning no erroneous characters should be ignored, `codecs.replace_errors`, meaning all erroneous character should be replaced

9.1.3 Flaws with the parsing

While this way of parsing is functional, and will allows us to got a workable version of the email archive into solr, it is far from perfect. The most notable issie is the fact that give a plan text message, we will get the entire payload, minus the attachement. This means that we willl potentially got a lot more than we actaully want, as all html will be included, if the email is just one in a long chain of messages, they wont be formatted in anyway beyond what the email client does by deafult. This again means we can end ub with very cluttered emails that are hard to read, simpy due to the formatting. This could however proove to be an advantage in machiene learning, assuming a correspondent sticks to one specific layout, it may be used to identify their other messages, or even figure out if one person is using several adressess.

As prevously mentioned, i will not be using any specific solr schema at the moment, this means that all info passed onto solr has the type "string", and while solr does automatically try to determine the type of the data beeing sendt to it, it is most likely to defaul to string, something that is the case for all of our document field (as we shall see later). This also means that fields like "Date:" do not have a standarized syntax, and rely on whateven was in the field at the time of its extraction. This also means that searching this field will return ureliable results

The get() method used to extract the headers is also far from perfect, and will frequently return empty values even if the requested header is both present, and non-empty. This could be solved by doing some simple text matching and identifyng the fields that way.

9.1.4 Solr schema

Fields and tokenizers

each field need a specific tokenizer the token produced will be indexed and used to search the data

tokenizers in solr can do more than just split the words apart for us, case folding is one of such things

Pre processing Before we can start tokenizing we firstly need to establish what kind of pre processing a field needs, if any at all

Date this is definitely a field that should be processed into one unifier format across all email, as it will make it easier to search, lucky for us Solr has a Specific "Date" field type with is own format, meaning that the date extracted from the emails needs to be processed to fit the solr format. This is something that needs to be done by the parser

Timezone This is very similar to the last case, witch the exception of Solr not having a field type for time zones. Needs pre processing in the parser

From As this is a case sensitive according to `https://tools.ietf.org/html/rfc5321#section-2.3.11` we cannot case form this field, it shouød however still be considered as most email systems used today do not distinguish the addresses based on case

this applies to all field that contains email addresses, those being:

From Sender Reply-to To Cc In-Reply-To

Message-ID As this is a unique identifier of a message, that is case sensitive, it should be left alone as changing it would also change its meaning

Subject As this often is natural written text, written by a human being, it should be left alone as we can only assume that the author made the subject field the way he or she intended, it us not up to us to assume that this field needs any for of correction.

Mailing list As this field is filled out based on the filename of the file the message was residing in, there is no need for further processing here either, the IETF email archives follow the same convention when it comes to naming and organizing the mbox files File location Again this field is taken right from the system responsible for the storage of our files, no need for further processing

Content

I believe that this field falls under the same case as the subject field, we can only assume the content of the message is presented such as it is, because thi is the way the author intended for the message to be.

One thing to note here is that the python email packages are not perfect when it comes to extracting this field. We will often see barts of code or attachments make their way into this field in some capacity. While this does not make for an excellent reading experience, it does not make it unreadable, as these kind of errors almost always appear at the very end of a message, often separated from the actual content by several new lines.

We can also to some extent deal with the by using the correct tokenizer, this is something i will come back too.

Tokenizers

I will not present every single one of Solars tokenizer as this information is better acquired from the official documentation available here `https://lucene.apache.org/solr/guide/8_2/tokenizers.html`

What tokenizer in each field

Date ?

Timezone Standard tokenizer

All email fields URL amil tokenizer

Message ID keyword tokenizer

Subject whitespace or standard Content whitespace or standard

Mailing list Keyword tokenizer

file location standard tokenizer

9.2 Statistics

Number of files in the downlaoded text folder use command find /home/cezaryrj/IETF/ietf.org/mail-archive/text/ -type f — wc -l

this returns 77224 files

number of documents after parsing these sfiles is 2765548 mening we have 2765548 individual emails in the databse, asuming the parsing was correct

Chapter 10

Designing a full text database

In order to transform pure text into something one can make sense out of and search, it is first necessary to design a system that will allow for such functionality. Under normal circumstances, this system would essentially be a search engine, where tokens are indexed in a reverse index, and results are calculated based on, for example, similarity to the searched keyword or phrase. In a modern search engine, classification also plays a big role in both how tokens are classified, and how results are calculated.

In this case classification is something I will come back to, it is not needed for now.

As previously mentioned, i did not develop a new text database system, instead a program that already offers the functionality i needed was used. This program is called Apache Solr, it allows me to feed it data, given in a set format, and process it according to my wishes.

In order to make proper use of Solr, one first has to design its schema, the schema defines how the database will look, by that I mean what fields will it have , what kind of data is stored in each field, and how should data in each field be processed. The schema also defines the set format in which input data should be given. This format is referred to as a document, each document has fields which in turn have their own rules that can be adjusted as needed.

This means that a “layout” for a document needs to be defined first. This is not strictly necessary as Solr will automatically add new fields and adjust their rules according to what it sees fit for that field. While this is not very useful for this project, it is there, and it means that there will be no crashes if a field that is not defined in the database schema is encountered. However, for ease of use, it is possible to first, upload a document, with the desired layout, and then manually adjust the settings for each field in the schema file. The schema file can only be manually changed when Solr is not up and running.

10.1 The Document format

As previously stated, a layout for a document is needed, in this case, a document is meant to represent exactly one email, in its entirety. This does not mean all of the headers, rather, all headers that have been chosen as useful to extract data from, as well as the massage body. Attachment will not be included as they take up a considerable amount of not only storage space, but visual space as well , making the e-mail difficult to read,something that goes against the goals of this project.

For further research one could attempt to restore the document from their ASCII representation back into their original form, then parse them. That is however outside of the scope of this project.

Then, what exact information should be extracted from the emails ? As much data as possible, as long as the data is following the standard set by the IETF, or data that does not follow the standard, but can still be transformed into a useful format. Also, data that is is common in email messages, so while custom headers are allowed and the support for them exists, it is impossible to predict what they may be called, or what data they may contain, making it impossible to predict how said data should be processed, they will for that reason be excluded from this project, only headers defined in the most recent RFC describing the email format will be considered.
As not all headers are considered mandatory, there will inevitably be fields that are missing, yet are expected to be present, for such occasions a default value must be specified. Not all fields will accept the same value however, therefore a default value must be specified for each individual field. This is done by the parser, and not Solr. Since Solr simply omits fields that have no value, resulting in documents that will not be uniform. It is also easier to determine that a field is empty, if there is a default value in its stead, rather than the entire field missing. So while this is strictly speaking not necessary for the functionality of the database, it will make reading the documents easier for the end user.

Date (orig-date)

This field contains the date of sending, as described in RFC5322. It is also a mandatory field, and is therefore included in the document template. The data extracted from this field will be formatted to comply with Solr-s "date-field" type. This will allow searches based on date, such as, from and to a specific date.

The default value for this field, must be at least older than the oldest email present in the database. To make sure that this is the case, the default date has been set to 1. Of january 1900.

From

This is also a mandatory header and as such is included in the document template. One important thing of note about this header, as well as many other headers, is that it contains email addresses, and according to rfc5322, it can also contain the sender name, in addition to the address. Another challenge this, and many other fields present, is the fact that it can contain several addresses, and those again, can contain names. This in itself is not a major problem as the python email module has functionality that allows for easy extraction of this information. Solr however, has no concept of lists, outside of returning results that is. There is simply no way to tell Solr, that a list of something belongs in a field. There is a way to work around this however, since Solr has a reverse index of tokens, and uses that index to answer queries, what really matters is how the data is given is being tokenized. This means that if we were to transform a list of names into one single string, with the name being separated by whitespace the resulting tokens would be the names, meaning they would be searchable in that field. The problem of first and last names, which are normally separated by whitespace can be easily solved by replacing that whitespace with an underscore. Using this workaround means that for every field that contains, or may contain and email address and name pair, there must be an additional field, processed by a whitespace tokenizer, these fields will be named as follows:

(original name of field)-name

Additionally, another field called

(original name of field)-address

Will be added, and containing all of the addresses found in the “From” field. To make sure that no data is lost during processing, the “From” field itself will remain untouched.

- Sender
- Reply-To
- To
- Cc
- In-reply-to

Those are fields that contain email address and name pairs, therefore they will be processed in the same way the From field is. These are not a mandatory fields, as such, absence of data in those fields will not be treated as an error.

Message-id

This is a mandatory field that uniquely identifies a message on a global scale. They are very useful for tracking down specific messages. This field will for this reason not be processed in any way aside from removing preceding and succeeding whitespace. Absence of data in this field will count as an error.

References

This field contains the message id of a referenced message.This is not a mandatory field, but still a field with a unique key, and should therefore not be processed in anyway aside from the one mentioned above.

Subject

This is what is commonly referred to as an "unstructured field", meaning there is no concrete pattern or standard as to how data in this field should look. It can be anything from one word to a whole sentence, or just some completely random characters. With that being said it is probably safe to assume that most people would use this field in a sensible way, that being to highlight the motive or goal that lead to the emails creation. This field should therefore be processed in the same way as the payload of the message, aka as pure text. This is not a mandatory field and the absence of data in this field will not be treated as an error.

Comments

This is also an "unstructured field", as such it should be processed in the same way as described above.

Payload

This again, an unstructured field commonly used to contain the actual text of the email, as will therefore we processed like the 2 fields described above.

10.2 Tokenizers

I have already briefly discussed tokenizers and their effect on how results are calculated. Therefore in order to ensure that a search on a given field yields the expected results, fitting tokenizers for that field must be chosen. Luckily Solr has a set of tokenizers built right in. The tokenizers are specified on a per field basis, meaning that each field in a document can be processed differently without it affecting any of the other fields.

The tokenizer for each field is specified in a database schema file, any changes made to a field's tokenizer will require a re-upload of the entire database to take effect.

From

As previously mentioned this field, at least in this database, will contain unprocessed data, the same string extracted from the mbox file, is the same string that will occupy this field in the database. In order to make this field useful however, Solrs standard tokenizer will be used.It treats "@" as a delimiter, meaning all email addresses will be split into several tokens, this is not a problem, as the previously mentioned address field will already contain these.

Solr standard tokenizer = solr.StandardTokenizerFactory

From-name

As this field contains what is essentially keywords, Solr-s whitespace tokenizer will be used, as previously mentioned. Given that the separating whitespace between first and last name has been replaced with an underscore during parsing,the tokenizer will produce tokens that are the actual names, but with an underscore. This also means that it will be necessary to use underscore instead of space when searching this field.

Solr whitespace tokenizer = solr.WhitespaceTokenizerFactory

From-address

As i have already established, this field will exclusively contain email addresses, for this reason Solr's email tokenizer will be used. The same will be true for other fields that contain exclusively email addresses.

Solr email Tokenizer = solr.UAX29URLEmailTokenizerFactory

The following field will be processed in the same way as the From field :

- Sender
- Reply-To
- To
- Cc
- Bcc
- In-reply-to

Message-id

As this is a unique identifier, and should not in any way be changed, the keyword tokenizer wil be used. The keyword tokenizer is exactly what the name implies, and produces a token that is exactly the same as the field's content, in other words, it does nothing to the data given.

Solr keyword tokenizer = solr.KeywordTokenizerFactory

References

This field is no different than Message id, and will be treated in the same way.

Subject, Comments and Payload

As these are all fields with the same characteristics, they will be tokenized by the same tokenizer. These are all unstructured fields, meaning there is no definitive answer as to how these fields should be tokenized, therefore, Solr's standard tokenizer will be used.

Solr standard tokenizer = solr.StandardTokenizerFactory

Add a final template here

10.3 Can the python email module be used ?

Before we deem the mailbox module appropriate for our use, we first need to make sure that it even tho i follows the RFC2822, which has been long obsoleted by newer rfc, it can be still used.

In short, i can only use the python module, if a message that is compliant with rfc6854, which is the most recent rfc regarding the email format at the time of writing, is also, at the same time compliant with rfc2822. At least that would be the assumption.

It is important to note that they do not have to be 100

To give an example, let's say we have a message, where the "From" header looks like this

"From: test.com"

This is not RFC2822 compliant way of creating and filing a header field, the python module however will still return "test.com" from the "get("From")" call, even though it is not compliant. If that is the case, what part of the message needs to comply with the rfc2822 ?.

First of all, all kinds of separators need to comply, as we are extracting many messages from a single file, the boundaries of a message need to be 100In that case, what boundaries occur in an email message ? According to RFC2822, inside a message, there is only 1 boundary, that being the boundary separating the headers from the body.

From the rfc2822 text :

> *"A new line that separates the headers from the body A message consists of header fields (collectively called "the header of the message") followed, optionally, by a body. The header is a sequence of lines of characters with special syntax as defined in this standard. The body is simply a sequence of characters that follows the header and is separated from the header by an empty line (i.e., a line with nothing preceding the CRLF)."*

Source: `https://tools.ietf.org/html/rfc2822#section-2.1`

There is no mention of any changes to this rule in later rfc-s, it is therefore assumed that it is still in effect at the time of writing.

Secondly, all names of headers need to comply, fortunately, names of headers we are concerned with have remained unchanged through the years. Some older emails may lack the newer headers, but that is to be expected, and the parser will be able to handle such cases.

Now we know that the changes made in newer rfc-s do not get in the way of our works.There still do remain older standards however. The IETF is an old organization, and email from ever before 1980 can be found in their database.

The earliest rfc that specifies any form of email format,that exists in the IETF-s database, is rfc 561, published on september 5. 1973, and lays the fundamentals of how a modern email message should look like. Most importantly, it specifies what the headers should be called. What it does not specify however, is which headers are required to be present. This means that I cannot determine with 100However, I believe that it is safe to assume that in order for a message to be proppery transmitted, certain information still needs to be provided, such as the recipient address and the sender address.

Another important thing to note is that the rfc specifically states that the case(upper/lower) of headers does not matter. The python “message” module, matches headers with no regards to upper or lower case, which is very convenient, it also has a method that returns a list of all fields with the same name. If any email is found to have several fields with the same name, the contents of those fields will be appended, and processed as described earlier.

In short, as long as the boundaries, and names of headers in a given email are correct, the mbox module will be able to properly parse it, regardless of the rfc. Should the email module be used on a given field however, then that field must conform to the newest RFC. Therefore the email module should be used cautiously.

10.4 Mbox format

To better understand the task at hand it is wise to first understand how to approach it, and while email messages are the goal, it is not email messages that are provided by the IETF. What can be found on the IETF-s website are their mbox files, an mbox file is a file containing messages sent to a certain mailbox, as the name implies. The first step in extracting the emails, is converting an mbox, back into some readable representation of emails. This is a simple task as mbox files are emails that are concatenated to each other and stored in pure text, meaning the only challenge is determining the boundaries of a message. It is important to note that the email will be stored in the same order they were originally received, meaning that multipart MIME messages may have their parts spread across an mbox file, making their reconstruction a non-trivial task. By definition, each message contained in an mbox file starts with “From” a space, and the sender’s email address and date, this is also referred to as an emails envelope. The envelope is not a header however, each message should, in addition to the envelope, also contain headers required by the rfc standard.

Example :

From MAILER-DAEMON Fri Jul 8 12:08:34 2011

From: Author ¡author@example.com¿

To: Recipient ¡recipient@example.com¿

Subject: Sample message 1

In order to avoid errors, lines that start with “From” but are not the beginning of a new message, should have a symbol, usually “¿” added at their beginning. The end of a message is usually signified by an empty line, but the start of a new message, as well as the end of file have also been used to signify the end of a message.

10.5 Error detection

As I discovered while attempting to parse the email archive, not everything in the archive has been done according to specification. This led to several problems, first of all, the python mailbox module assumes that any given email, follows the specification given by the IETF. It also assumes that that a given mbox file has been correctly formatted, with proper boundaries between the messages.

As I quickly found out, far from all files, and messages contained in them, are free from errors.This often leads to missing headers, as a result of eather misnamed headers, or erroneous boundaries within the mbox file. Resulting in the headers of a message, being interpreted as belonging to its predecessor,and the body as its own message.

This issue applies to both of pure text messages, as well as MIME messages. Text messages however seem to be more resilient against errors, as they are sent in one piece, unlike the MIME messages.

MIME

There are 2 main types of email, MIME, and non MIME messages. MIME stands for Multipurpose Internet Mail Extensions, and is essentially an extension to the normal text based messages. It allows for video, audio, images and even applications to be displayed and run as a part of the message. The downside to this is that the message body will often exceed its length limit, causing it to be split into several parts automatically, then re-assembled back into one message on the receiving side.

As one may imagine, this is a popular source of errors, as the MIME messages are split up into several parts, there is no guarantee that these parts arrive in the correct order, or arrive at all. This seems to be the second cause of missing headers, since, if a “rouge” part arrives by itself, it will be missing the headers, as it only carries information identifying which MIME message it belongs to.

ADD EXAMPLE HERE

The python email module does not make any attempt at correcting this issue, it expects all messages to be in sequence. Another issue I ran into is repeated messages, what i mean by this is parts of a MIME messages arriving several times. On the other hand, there are MIME messages where a certain part never arrived.

The third and final cause is data corruption, there is unfortunately no easy way to automatically determine this as far as i am aware . If any of the following headers are missing, an error report will be generated :

- From:
- To:
- Date:

10.6 Error correction

Find out how mime messages are ordered
Boundary spec may be missing

10.7 The error report

Given all this information, what can we really tell about an email. And what should be included in the error report ?

1. The email position in the mbox file,unfortunately , the mailbox module does not track line number, making it impossible to know exactly where in the file this message is located. Using message ID-s is not something we can use eather as they can be missing. This means that the only way of tracking down a specific message is to count how many messages come before it in a given mbox file.To make this easier ,a program that automatically prints message number n, in file f fill be provided.
2. Which file the message resides in
3. The type of the message (MIME or not MIME)
4. What information the the parser was able to extract
5. The message itself, in its raw form

Treating and repairing the data While it is important to make as much as possible usable to us, with the current parsing, less than 1

Even if an attempt at fixing these messages would be successful, there is no guarantee that they would provide any useful data, as some parts may simply have been lost in transmission.

For this reason i have decided that dedicating time to repairing such a small amount of messages, is simply not worth the time at the.