# 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 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 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 lemmetizing 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 also 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 messanges, 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

gdfgdfgd

## 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