Unique Channel Email System

Thesis for the Degree of Master of Science by

Milko Balakchiev

University of North Texas
Computer Science and Engineering
August 2015

Song Fu Major Professor	Bill Buckles Committee Member	Hassan Takabi Committee Member

Balakchiev, Milko. Unique Channel Email System. Master of Science (Computer Science), August 2015, 43 pp., references, 18 titles.

Email connects 85% of the world. This paper explores the pattern of information overload encountered by majority of email users and examine what steps key email providers are taking to combat the problem. Besides fighting spam, popular email providers offer very limited tools to reduce the amount of unwanted incoming email. Rather, there has been a trend to expand storage space and aid the organization of email. Storing email is very costly and harmful to the environment. Additionally, information overload can be detrimental to productivity. We propose a simple solution that results in drastic reduction of unwanted mail, also known as graymail.

ACKNOWLEDGEMENTS

I would like to thank the University of North Texas for helping me obtain a Master Degree. Especially my advisor, Song Fu and committee members Bill Buckles and Hasan Takabi. I have gained invaluable knowledge associating with you and completing your courses. Additionally, I am very appreciative of Stephanie Deacon and Sally Pettyjohn for the tremendous help with all things administrative.

Lastly, eternally grateful to my family and wife for being willing to live the student life along with our two children.

Table of Contents

INTRODUCTION

Email plays an important role in the majority of the worldwide population. It has become an integral part of our communication. We rely on email to get in touch with someone where phone may fail. It seems less intrusive than a phone conversation but more universal than text messaging.

183 billion emails are sent/received daily. About 70% of all messages, or 128 billion of them are spam (Sara Radicati, 2013). Fortunately, companies like Google and Microsoft have figured out how to detect more than 97% of that spam with less than 1% false positive rate. This is accomplished by sophisticated Bayesian classification machine learning algorithms. Despite the tremendous success rate of such precise classifiers, any percentage of false positives can have severe consequences if the email send to the spam folder happens to be important, such as a job interview. In other words, at 1% false positive rate, 128 million emails never get delivered.

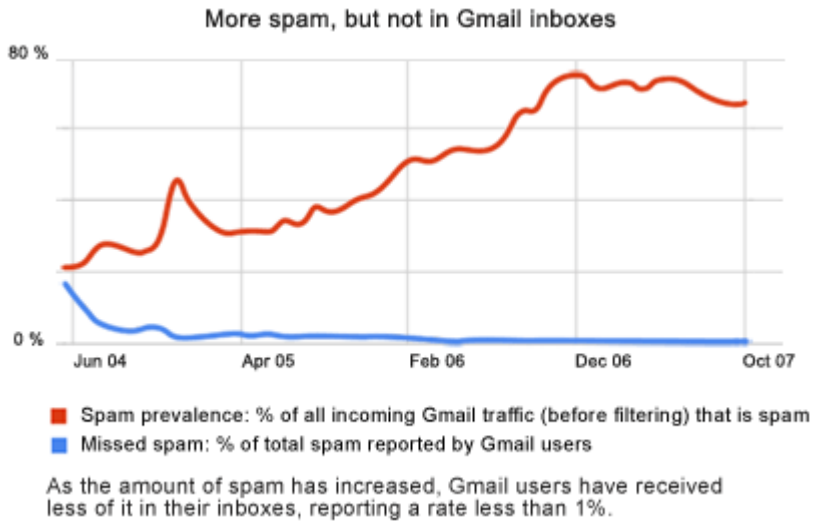


Figure 1. Gmailblog.blogspot.com

In addition, a simple experiment show how easy it is to bypass a spam filter in Gmail. First, open a new email account, then send a normal looking email to a Gmail account. If the email is not suspicious, it will be delivered to the user's inbox. Once that has taken place, any further communication with that contact will not be filtered. In fact, in our simple experiment, once the non-suspicious email was received in the inbox, we purposefully reported it as spam. We then copied and pasted 100 of the most common spam keywords into a new email message and sent the email to the same recipient. Surprisingly, the email made it into the inbox without any resistance. We verified that if a message is sent for the first time with the same 100 spam keywords, it is detected as spam, however if a message from a sender was judged to be safe at first sight, it is no longer subject to spam filtering. This may explain why some emails, that are obviously spam still make it into our inboxes. This opens the possibility that once we have signed up for a newsletter at a website, subsequent emails from the same sender will not be filtered.

Most, if not all email users have separate email accounts that they give out in different circumstances and most email boxes are cluttered by massive amounts of messages that are of little interest to the recipient. We estimate that as much as 80% of all non-spam email is of little interest to the user. This email is known as graymail. Graymail is email that users knowingly or unknowingly subscribed for but have no interest in reading. Graymail can take hours to manage, costing businesses and individuals valuable resources. Graymail can drown out wanted mail in piles of unwanted mail. With the advent of mobile technology where notifications are sent with the arrival of each new email message, the level of distraction is magnified. Many users check their email every 15 minutes or 100-plus times daily but are often left disappointed with every irrelevant email that shows up in their inbox. Graymail can be a distraction from more productive

activities. Intel report found that the cost of lost productivity from email overload for large firms with more than 50,000 employees is $1 billion a year. (Spira, 2009)

Users' circumstances change rapidly. A user may be searching for a car, job, scholarship, vacation location for a short duration but email newsletters and subscriptions last until the user unsubscribes. Currently, users pay for free website access with their email accounts, which translates into time and attention. Often the benefit of accessing a website is disproportional to the cost receiving hundreds of emails.

Below is a table that analyzes the top 30 sender in a typical gmail account from January 2014 to December 2014, sorted by most-emails-received from a contact.

Table 1 Top senders in a typical inbox

Rank	Sender	Total	Read	Open Rate
1.	Groupon <noreply@r.groupon.com>	815	12	1.47%
2.	Simply Hired Alerts <alerts-noreply+artmilko=gmail.com@jobs.simplyhired.com>	339	14	4.13%
3.	Personal Contact	333	333	100.00%
4.	"CommunicationsJobs.net Alert" <alert@email4-beyond.com>	263	2	0.76%
5.	Technology Job Insider <info@mail.technologyjobinsider.com>	193	2	1.04%
6.	Plano <PlanoTXFreecycle@mod.freecycle.org>	188	2	1.06%
7.	Godiva Chocolatier <godiva@email.godiva.com>	161	4	2.48%
8.	LinkedIn Connections <connections@linkedin.com>	142	4	2.82%
9.	"Brainsports.com" <artmilko@gmail.com>	141	86	60.99%
10.	UNT Announcements	136	58	42.65%
11.	Quora Digest <digest-noreply@quora.com>	122	17	13.93%
12.	Micro Center <eNews@email.microcentermedia.com>	119	2	1.68%
13.	Kate@sologig.com	115	3	2.61%
14.	Google Calendar <calendar-notification@google.com>	110	4	3.64%

15.	"CommunicationsJobs.net Job Match" <JobMatch@email4-beyond.com>	108	1	0.93%
16.	"Sologig.com" <JobRecommendations@alerts.sologig.com>	106	0	0.00%
17.	LinkedIn Updates <messages-noreply@linkedin.com>	103	6	5.83%
18.	Plano <PlanoTXFreecycle@mods.freecycle.org>	100	4	4.00%
19.	Today on Twitter <info@twitter.com>	97	7	7.22%
20.	Twitter <info@twitter.com>	95	6	6.32%
21.	LiveChat <support@livechatinc.com>	71	3	4.23%
22.	Citi Cards <citicards@info9.citibank.com>	69	3	4.35%
23.	Milko Balakchiev (emails to self)	66	47	71.21%
24.	Personal Contact	60	60	100.00%
25.	LinkedIn <messages-noreply@linkedin.com>	58	12	20.69%
26.	Personal Contact	57	57	100.00%
27.	Personal Contact	57	57	100.00%
28.	Personal Contact	55	55	100.00%
29.	Facebook <notification+5j1p35ki@facebookmail.com>	53	11	20.75%
30.	SkillPages Team <no-reply@skillpagesmail.com>	52	2	3.85%
		Total	**opened**	**open rate %**
		4384	**874**	**19.94%**

There emerges a clear pattern that open rate is a good indicator whether an email is graymail or important. Personal contacts frequently exhibit a 100% open rate. After taking the same data and sorting it by open rate, the resulting table contained *only* personal contacts in the top 30 rows. Another indicator is the quantity of emails. We observe that often graymail generates a large quantity of emails which points to the claim that about 80% of all email in an inbox is graymail. We came to the same conclusion with several accounts. Even organized individuals have less than 20% open rate.

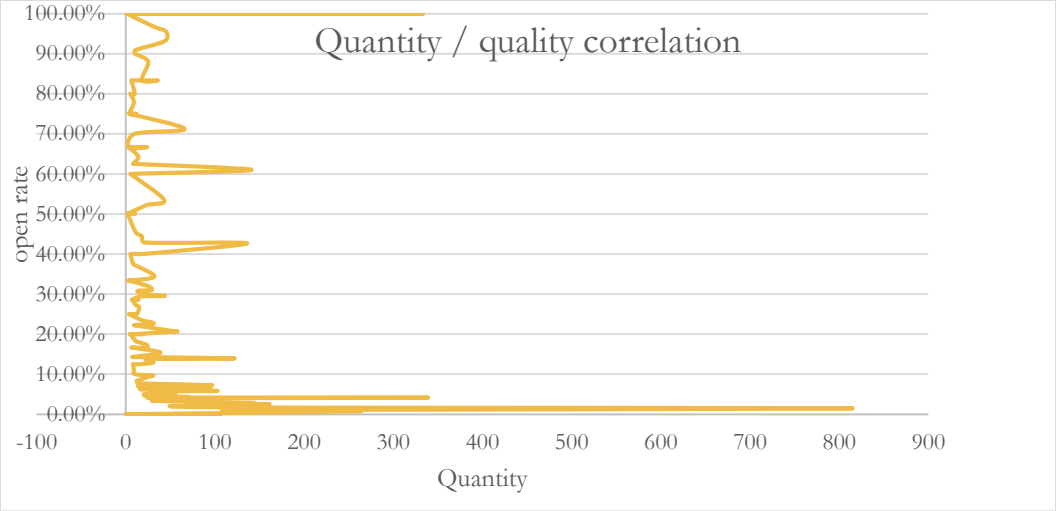


Figure 2 Quantity and quality correlation

Google has taken some steps in dealing with the problem of graymail with the introduction of priority inbox which offers the ability to sort email among different tabs such as primary, promotions, social and updates, as well as plus addressing. While this approach helps to sort your email into categories, it does little to stop unwanted graymail. According to Google, one has to create a filter for unwanted messages or unsubscribe by hunting for subscription information on the sender's website, which may not always honor removal requests or do so in a timely manner, or worse yet, there is no option to unsubscribe. Below is a quote from the google support blog outlining the instructions on how to filter a message. Notice that there are six steps involved:

"While you can't currently block emails from specific addresses or domains, you can set up a filter to send those emails directly to Trash.

1. Open Gmail.
2. Check the box next to the email you'd like to create a filter for.
3. Click More.
4. Click Filter messages like these.
5. Fill out information about the emails you want to filter. Click Create a filter with this search at the bottom-right.

6. Check the box for what you'd like to do with messages that fall into that filter. If you'd like them to get sent to Trash, check the box for Delete it." (Google Inc., n.d.)

Google's approach to dealing with email overload is to offer more storage space costing millions in storage, power consumption, network traffic and server equipment. Such an approach is wasteful and unsustainable on the long run.

We have developed a novel email system that tackles the problem of graymail by introducing a unique message channel for each contact with a simple on and off button. The anatomy of such an email address contains 3 parts; The user string, which is constant in every email, the domain section is also constant, which is the string after the @ symbol in an email, and the customizable portion which represents the recipient of the email. The latter portion can be a random string or in more legible format. For example, messages from facebook could arrive at facebook_user@domain or at <random string>_user@domain. The reason for such an option of obstruction is that it will prevent spammers from guessing a valid email address, although such attempt are unlikely. Important consideration are that all emails be recognized as valid email addresses and that the distribution of a new email is convenient for the user. Our implementation attempts to solve the most common case of providing an email, which is an online environment but other scenarios are considered in the discussion section of this paper.

Such an email system can go a long way in controlling graymail, spam and inform the user about who leaks their email address to advertisers. Furthermore, such a mechanism offers valuable protection against phishing attacks, attacks in which the email sender asks for sensitive information by pretending to be someone else.

Another benefit of implementing a unique channel of communication on per-contact basis is that email messages can easily be classified in folders verses manually creating a folder and labelling it, a common practice in Outlook and Gmail. Having the messages sorted into folders makes it easy to delete in bulk and to implement timing heuristics. Messages from a certain recipient can be set to self-destruct after certain amount of time of sitting inside the inbox in unread state. A user may only be interested in browsing through the subject line of the message but not interested in reading the actual message. A common occurrence for such messages maybe with promotional content from vendors like Groupon and LivingSocial, which appear to be the top generators of graymail. The user may only be interested in one percent of the advertisements but not entirely disinterested to cancel subscription to the service entirely.

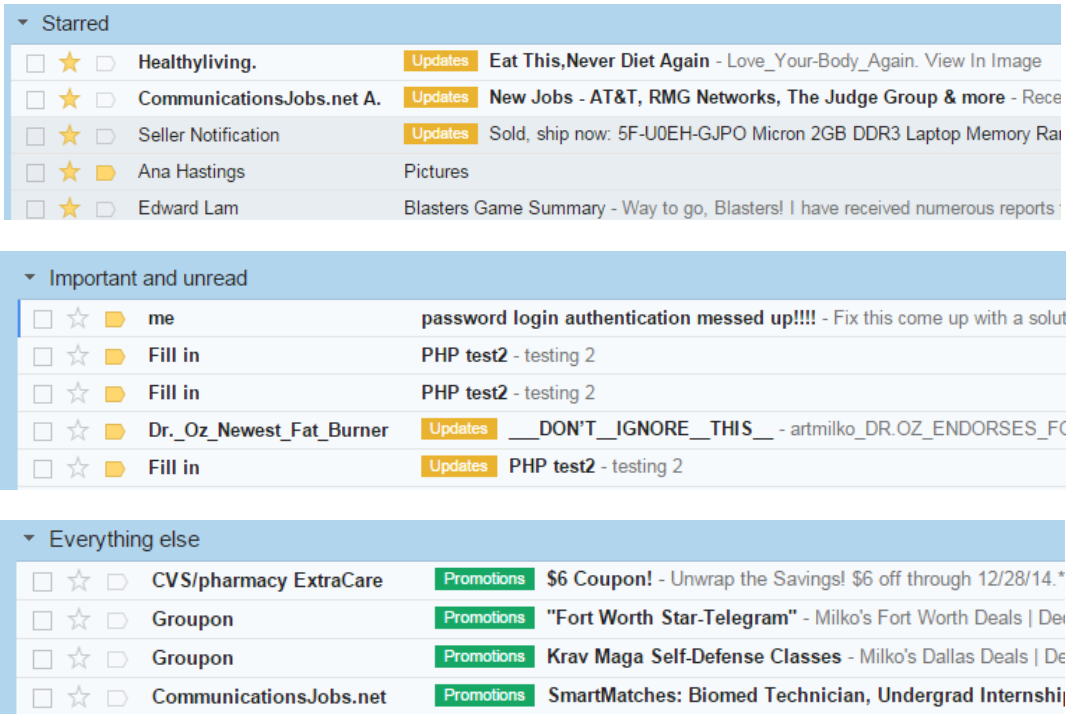
While such deletion timer can be manually set by the user, our implementation "studies" the user's behavior in reading emails from a particular recipient and determines what to do with emails arriving on the specific message channel. Once a pattern is determined, automatic deletion of unread messages is triggered after residing in the inbox for certain amount of time. If no emails are opened for long period of time, the user is prompted with the choice to stop receiving messages through that particular channel. Once the channel is blocked all messages from the recipient can be deleted in bulk with a single click.

In our testing, such an email system has proven very efficient against spam and graymail. Our future work involves additional implementations that would further streamline the process of bacon-only email.

In 2008, Gmail introduced Aliases in the form of plus addressing, in an effort to offer users a better way to track incoming email. Gmail explains that this is the structure of the alias: your.username+any.alias@gmail.com. (Google Inc, n.d.)The major shortfall being that by removing everything after the plus sign, one can easily figure out the original email address, rendering the feature useless. Microsoft soon followed in 2011 with plus addressing implementation as well as allowing up to 5 aliases or unique email addresses. (Microsoft, n.d.)

In 2010, in another attempt to address the issue of information overload, Google introduced a feature called Priority Inbox. Priority Inbox separates email into three sections; Important and unread, Starred, and Everything else.

Figure 3 Priority Inbox



They took a machine learning approach which calculated a probability that the email will be of importance.

They looked into several features and grouped them in to the following categories: Social features, measure the degree of interaction sender and recipient, Content features to asses terms in the header or content of a message that can predict the likely hood of the email being opened by the user. Next, they look at the Thread features monitor the user's interaction with emails from a particular recipient thus far. Finally, they look at the Label features which are used to gather information about the importance of an email by the labels the recipient used. The formula used by google to predict the probability that the user will interact with the mail within T seconds of delivery is

$$p = \Pr(a \in A, t \in (Tmin, Tmax) | \mathbf{f}, s);$$

where a is the action performed on the mail, A is the set of actions denoting importance, t is the delay between delivery and the action, f is the vector of features, and s indicates that user has had an opportunity to see the mail. Tmin is less than 24 hours and Tmax is measured in days. For the model they use simple linear logistic regression to allow scalability. They use a form of transfer learning which uses the sum of the global model and user model log odds:

$$s = \sum_{i=1}^{n} f_i g_i + \sum_{i=1}^{n+k} f_i w_i, \quad p = \frac{1}{1 + \exp^{-s}}.$$

Here n represents the number of features, k additional user specific features. G is the global model weight while w represent how different the user is from the global model.

Finally, the models are updated for each mail with weight for the ith user by following equation:

$$w_i \leftarrow w_i + f_i \frac{\operatorname{sgn}(e)\max(|e| - \epsilon, 0)}{\|\mathbf{f}\|^2 + \frac{1}{2C}}$$

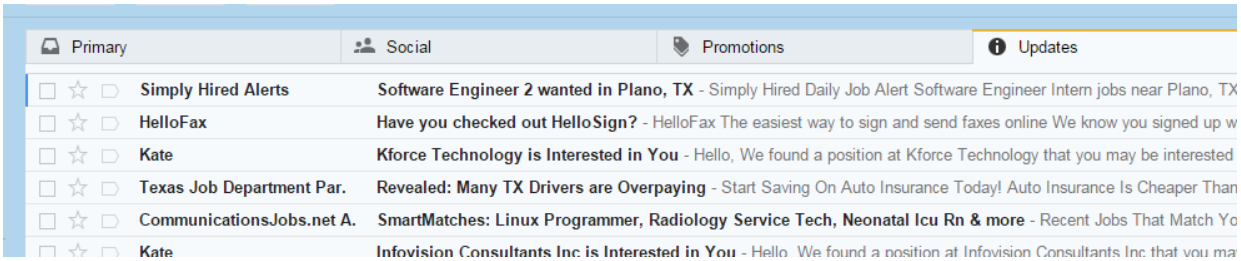
Where e is the error, C is the regularization parameter with ϵ being the hinge-loss tolerance.

As a result of this approach, they claim that their employees spend 6% less time on reading mail and 13% reading unimportant mail. (Douglas Aberdeen, 2009)

It is very difficult to judge which emails are important on individual basis. The emails that priority inbox believes are important are very unimportant to me, and include some spam. The layout disrupts the timeline of emails and is not user friendly.

As Priority Inbox failed to gain momentum, Gmail introduced a new inbox featuring tabs which attempt to sort emails into categories. The new inbox can sort incoming mail into Primary, Promotions, Social, Updates, and Forums.

Figure 4 Tabbed Inbox



The tabbed inbox attempts to address the issue of information overload by better organization. Although, they do not disclose their approach in implementing the tabbed inbox, we assume that they use similar model to the priority inbox and spam detection. Clever marketers know how to avoid getting classified in the promotional tab by addressing the

recipient by name, avoiding images and multiple links, as well as not selling directly in the email body. (Guy, n.d.)

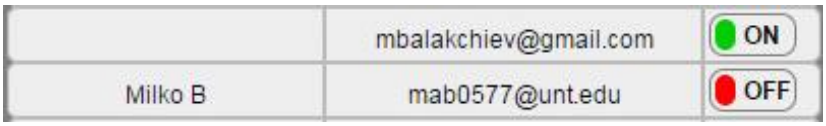
While tabbed inbox has done wonders to organize email into categories, it does little to reduce the flow of information.

PROPOSED SOLUTION

We propose an approach that not only organizes but controls the amount of information. For the mail system to work we believe it needs to be easy to use, be in complete control of information inflow. The key components of our mailing systems are;

First, one-click unsubscribe as this is the main source of information overload.

Figure 5 One Click unsubscribe

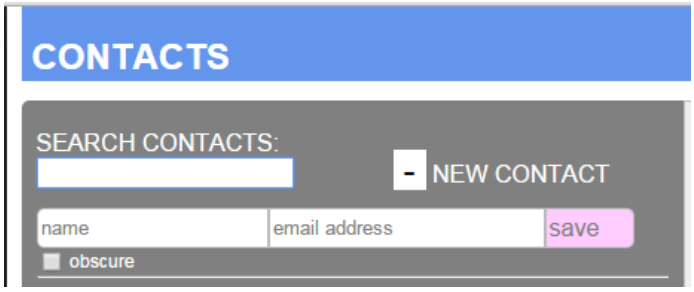


Second, as it is difficult to judge which emails are important on individual basis we have implement a unique channel email system that assigns a unique string to each contact allowing for easy tracking and control of information. This way we can treat all emails arriving at channel with same level of selectiveness without having to analyze each email.

Figure 6 Unique From address field on per-contact basis

When visiting a website that requires your password, simply add the alias to the database, and provide the email address to the website requiring it. When adding the name of the contact, it can be added either by name or email address.

Figure 7 Adding a new contact



The obscure checkbox can be selected for further disguise. The algorithm for adding a contact is outlined as follows:

```
1.  if only name is provided OR only email is provided by user
2.
3.      if obscure is selected
4.          Perform an md5 hash function plus salt on the string
5.          and append to _username@domain
6.      else
7.          append as is to _username@domain
8.
9.  else if both name and email are provided
10.     if obscure is selected
11.         Perform an md5 hash plus salt on the email
12.         and append to _username@domain
13.     else
14.         Append as is to _username@domain
```

The reasoning behind the obscure feature is to prevent spammers from potentially predicting an existing email address. For example, if it is predictable that 80% of all internet users shop at Amazon, an intruder can safely predict that amazon_username@domain is likely a valid email address and can compromise the effectiveness of the email system. If obscure is selected, then the email will resemble something along the lines of

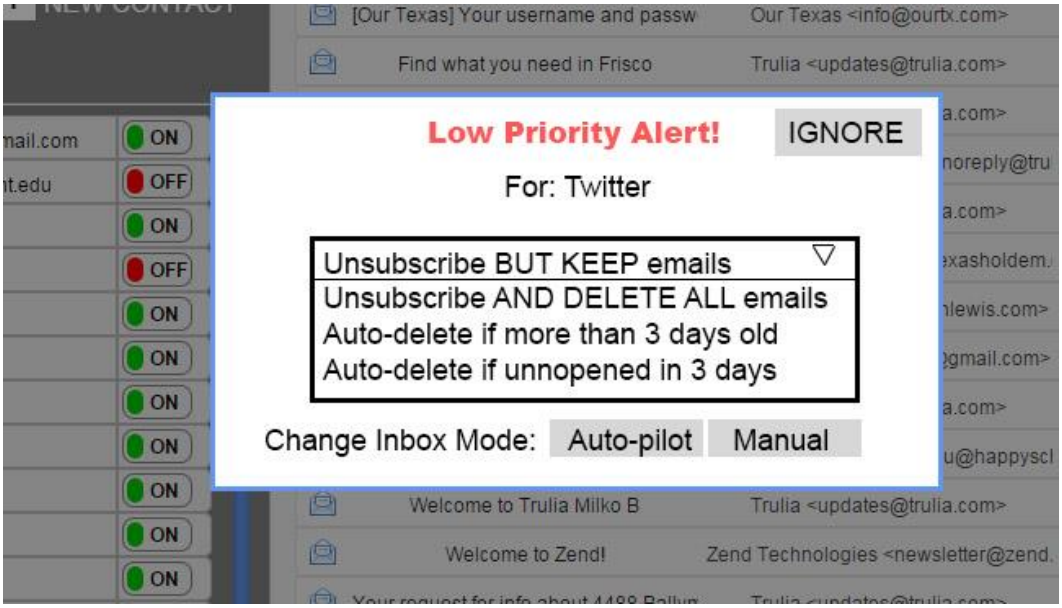
“6e720511a7b28_username@domain”. In addition, using a hash function ensures that the same predictable outcome can be accomplished but provides the benefit of appearing random. Even more importantly, using a hash function ensures that two different strings don’t get obscured to the same email address, a phenomenon known as collision. Collisions of an md5 hash functions are very rare (2^{64} hashes).

Another important consideration is the underscore in the email address. Google and Microsoft use “+” but after experimenting with several frameworks, the plus symbol could fail the validation test for a valid email address even though it does not violate RFC 5322 format(2). A plus in an email address is very unusual and websites that collect information with the intent to sell it may choose to block plus addressing to avoid poor quality and being tracked or filtered. In out implementation, it is very important to make the process as smooth as possible, therefore we used the underscore symbol, which is never flagged as invalid in an email address.

Finally, to ensure that the system prevents unwanted emails, the email username@domain is disabled. That way, if someone tried to cover their tracks and removed the unique sting in front of the username and tried to access a general inbox, they will get an invalid email response.

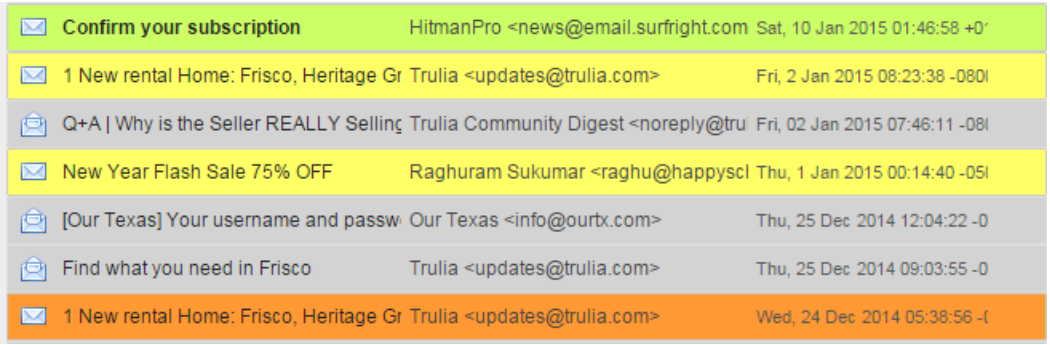
Third, in addition to the manual unsubscribe feature, we have implemented a tracking mechanism that monitors the traffic through each unique channel. If the user shows a habit of not opening emails in a particular channel, they will be presented with a prompt giving them the option either A) block emails from sender, with the option to delete all previous correspondence B) Auto delete emails from sender in x amount of time such as three days or two weeks.

Figure 8 Low open rate warning.



Further, the user can set the auto delete time manually on any inbox or even allow the email system to run on auto pilot. Auto pilot, automatically detects when emails from a particular recipient are not read very frequently, triggering auto delete in x days. To help the user visualize how "fresh" and email is, we color the line lime green for recent messages, light yellow for older messages and orange for messages about to be deleted. If the user has no time to read a message they are interested in, but it's about to get deleted, they can simply click on it. Doing so will set the "seen" flag and prevent deletion.

Figure 9 Color coded messages signal when deletion is approaching.



If emails are not opened at all for a prolonged period of time, the channel is automatically blocked by the system.

Figure 10 User is in complete control with the option of three modes.

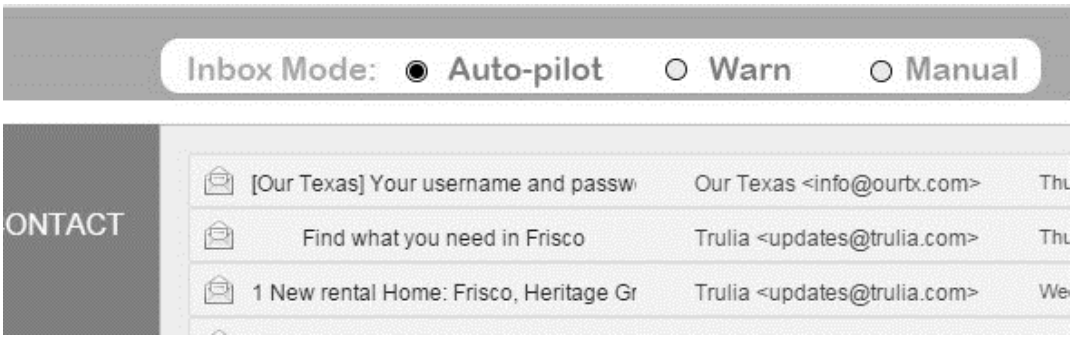


Table 2

Auto-pilot mode decision matrix	
Event	**Action**
Below 65% open rate within a month	Delete unopened emails after x days
Below 10% open rate within a month period.	Unsubscribe and delete all unopened emails.

The reasoning behind the auto-pilot threshold is based on the email analysis similar to the one in the chart provided earlier in table 1. We observe that all the graymail has 65% and lower open rate. The system will determine the average time it takes the user to open an email then use the upper bound, plus additional safety cushion before deleting emails. Our formula

resembles some of the features of the one used by Gmail except we don't consider other features so f is removed.

$$p = \Pr(a \in A, t \in (Tmin, Tmax), s);$$

where a is the action performed on the mail, A is the set of actions denoting importance, t is the delay between delivery and the action, and s indicates that user has had an opportunity to see the mail. Tmin is less than 24 hours and Tmax is measured in days.

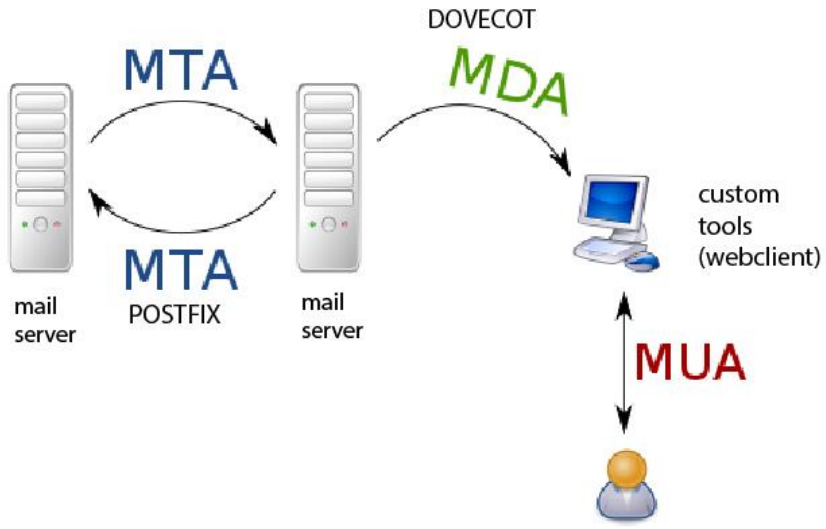
The 10% threshold for unsubscribing is also derived from analyzing email content. It appears that emails with less than 10% open rate are ones that we provided out of necessity and have no interest in reading anymore. These are the emails that have only the confirmation email opened and not much more after. Both of these values are subject to further tuning.

IMPLEMENTATION

To implement the email system we begin by setting up an email server. The operating system of choice is Red Hat Enterprise on Amazon EC2 micro instance.

To setup a server we need a Mail Transfer Agent (MTA), which servers to deliver mail among servers via SMTP. MTAs are also known as outgoing mail servers. In our case we used Postfix, an open source MTA. We also need a mail delivery agent (MDA), or incoming servers, to deliver mail from the server to the client computer. In our case we utilized Dovecot for this purpose. Dovecot manages mail upon receipt and sorts by user and folder. We communicate with Dovecot through Imap via a custom-built Mail User Agent (MUA). The MDA can communicate with the MUA via POP or IMAP. Since we are only using a web interface, using IMAP makes more sense as the email client and server need to be in synch. Communication with the cloud was established via WinSCP, and Putty SSH while database creation, management and alteration is accomplished through command line and PHPMYADMIN.

Figure 11 Components of the email server



On the client side, we used HTML, CSS, JavaScript and Ajax. While the communication with the server is accomplished with PHP. MySQL serves as the database.

The main database is called vmail and it contains fourteen tables. The table of interest in this implementation is the aliases which has fourteen rows as depicted in the figure below.

Figure 12 MySQL schema

	#	Name	Type	Collation	Attributes	Null	Default
☐	1	**address**	varchar(255)	utf8_general_ci		No	
☐	2	**goto**	text	utf8_general_ci		Yes	*NULL*
☐	3	**name**	varchar(255)	utf8_general_ci		No	
☐	4	**moderators**	text	utf8_general_ci		Yes	*NULL*
☐	5	**accesspolicy**	varchar(30)	utf8_general_ci		No	
☐	6	**domain**	varchar(255)	utf8_general_ci		No	
☐	7	**islist**	tinyint(1)			No	0
☐	8	**created**	datetime			No	0000-00-00 00:00:00
☐	9	**modified**	datetime			No	0000-00-00 00:00:00
☐	10	**expired**	datetime			No	9999-12-31 00:00:00
☐	11	**active**	tinyint(1)			No	1
☐	12	**contact**	varchar(255)	utf8_general_ci		No	*None*
☐	13	**cname**	varchar(255)	utf8_general_ci		No	*None*
☐	14	**hash**	varchar(255)	utf8_general_ci		No	*None*

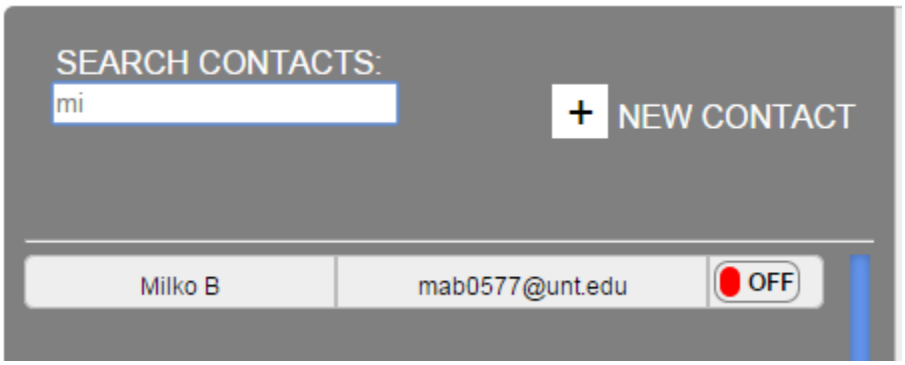
Dovecot initializes and maintains the core of these tables but our implementation introduces new columns to for the custom email system to function.

Visually, the website consists of three sections; the header, sidebar and the messages partition. The header section contains the links to the inbox and new mail, as well as the inbox mode options of Auto-Pilot, Warn and Manual. The side section contains three features; a search box, adding a contact area and a section where the contacts are displayed. The search box is unique in that it allows for queries to be queued to the server asynchronously. This is made possible by Ajax. Ajax allows communication with the server without the need to refresh the contents of the entire page. Upon releasing the key after typing a new character in the search box, the function showHint() is called. Control is transferred to the java.js file in the include folder, in the root directory. A new xmlhttp object is created. For older browsers such Internet Explorer 5 and 6, an ActiveX object is created instead. Upon ready status, a GET request is send

to another php file called contacts.php. Contacts in turn, performs a database search with a wildcard symbol before and after the string being sought after in both the name and email columns in the vmail table for the right user After which the parameters are passed back to the java function showHint() and the information is forwarded back to the contacts panel, only displaying the contacts that match any part of the name or email, along with the status of on or off.

If nothing is being typed in the search box a query is sent to the database, fetching all contacts and displaying them.

Figure 13 Search box with dynamic content



Next important feature is the add contact fields. It is positioned directly over the contacts to be more easily associated with the format. It is in collapsed state by default to avoid cluttering the interface. Once the plus sign is pressed, the show() function is called in java.js with the name of the two ids that it modifies, namely the id of the plus sign div and the div container holding the add contact form fields. Additionally, the name and email address fields contain a light gray text “name” and “email address” respectively. When the user clicks inside the field, the text is

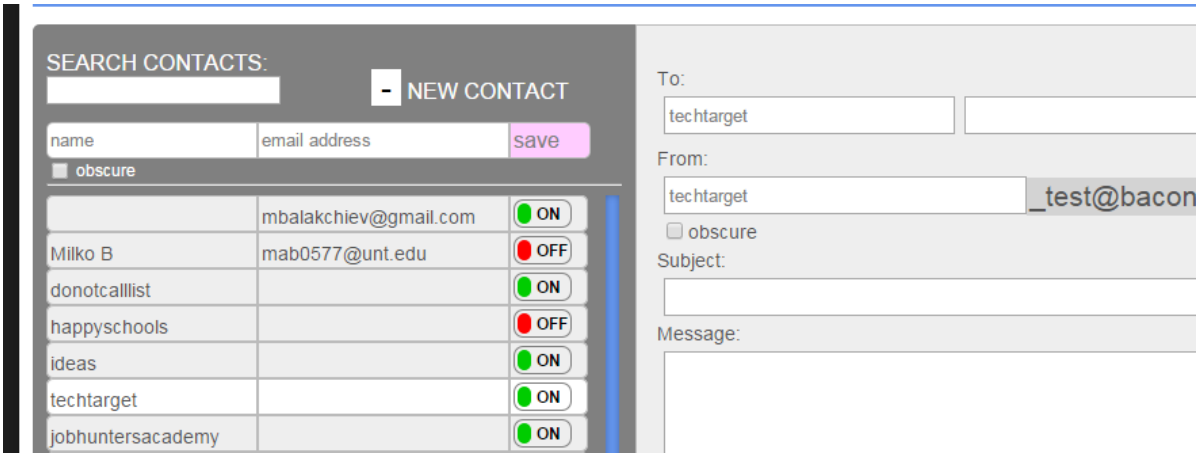
removed, if no text is entered, and the user clicks somewhere else on the screen, the text “name” and “email address” appear again. This effect is accomplished by the onfocus form field property in HTML. Upon submit the form content is sent for validation by Javascript function validateForm().

The validateForm() function receives the name and email address parameters. We then check the index position of the “@”symbol as well as the index position of the last “.”. At the end, we perform simple comparisons, to ensure that the position of the @ sign compared to the dot are logical.

Upon success, the form content is forwared to test.php. The form’s content is checked, if it contains name, email or both, and checked against the database, when it is written to the vmail table in the database as a new row and the user is forwarded to the contacts page again with the updated contacts content.

Finally, the contacts section of the side panel consists of a series of divs that resemble a table. Each row contains the contact’s name and email address and it’s on or off status as reflected in the database vmail table. For better legibility, the contacts have a gray background until the mouse hovers over a row, in which case it the background of the row turns white. If the user is composing a new email and a contact is pressed, the load() function is called in java.js. The load function parameters are the name and email address in and email address out, and it modifies the “to” fields of name and address as well as fills in the unique outgoing address. This step is very important as we do not want the responsibility of keeping track of which email is used by what contact, to fall on the user.

Figure 14 Auto load feature implemented with JavaScript.



If the user is looking at the inbox instead of composing a message, the contacts now become a link to a folder that contains only messages from that sender. To accomplish that, we utilize the imap_search() function, with the "TO" email as the parameter.

Every time an email arrives, the open rate is recalculated and stored in the database. If it falls below our threshold value, messages from that recipient start getting deleted after a certain period. We simply keep track by storing a running total of both read and unread messages in the database and calculate the percentage by dividing read by the total. Incoming streams which have a low open- rate receive a flag in the database, either the "timedelete" column is set to true or the active column is set to 0, which is the equivalent of unsubscribing from the newsletter.

Labelling the messages by a different color is performed by examining the difference between time of arrival and current time, something that is only done on low-open rate contacts. We can obtain the time of arrival with the help of the imap_fetch_overview() function which can provide us with the following information as necessary:

- *subject* - the messages subject
- *from* - who sent it
- *to* - recipient
- *date* - when was it sent
- *message_id* - Message-ID
- *references* - is a reference to this message id
- *in_reply_to* - is a reply to this message id
- *size* - size in bytes
- *uid* - UID the message has in the mailbox
- *msgno* - message sequence number in the mailbox
- *recent* - this message is flagged as recent
- *flagged* - this message is flagged
- *answered* - this message is flagged as answered
- *deleted* - this message is flagged for deletion
- *seen* - this message is flagged as already read
- *draft* - this message is flagged as being a draft

This is also how we acquire the “TO” information for the sort by contact in folders in the previous section.

The last panel of importance is the panel where email is composed or read. It is the frame on the right side of the screen, and is the portion of the email client that takes up the most room visually. This is where the emails are shown once received and where new emails are composed. To receive emails, we have chosen to display them in an accordion style. Initially, the messages are stacked, only showing the subject, sender and date. Once clicked, the email unfolds and shows the actual message. Multiple messages can be opened at the same time.

On the client side, the accordion affect for the message overview is accomplished by calling the toggle() function onclick while passing the id of the div, which is based on the mail id of the email. To obtain the email id, we open an imap stream and search for the email we wish in

the inbox. In our case we search for all email and then assign an ID to each, starting with one for the oldest email. The toggle function performs a simple if, else check on the div:

Now to get the body of the message is more challenging. A message body can have the following structure but it depends on the email client.

1 - Multipart/alternative headers

1.1 - Plain text message

1.2 - HTML version of message

2 - Inline attachment, etc

In Apple Mail the structure looks differently:

1 - Plain text message

2 - Multipart/alternative headers

2.1 - HTML version of the message

2.2 - Inline attachment, etc

Additionally, if a message has been forwarded, it will look something along the following lines:

1 - Multipart/alternative headers

1.1 - Plain text message

1.2 - HTML version of message

2 - Message/RFC822

2.0 - Attached message header

2.1 - Plain text message

2.2 - HTML version of message

2.3 - Inline attachment, etc (4)

The following is an example of the result of the structure of the simple email message: “This is a test message.“:

```
stdClass Object
(
   [type] => 1
   [encoding] => 0
   [ifsubtype] => 1
   [subtype] => ALTERNATIVE
   [ifdescription] => 0
   [ifid] => 0
   [ifdisposition] => 0
   [ifdparameters] => 0
   [ifparameters] => 1
   [parameters] => Array
      (
         [0] => stdClass Object
            (
               [attribute] => boundary
               [value] => 001a113ed5f297171a0508650a01
            )

      )

   [parts] => Array
      (
         [0] => stdClass Object
            (
               [type] => 0
               [encoding] => 0
               [ifsubtype] => 1
               [subtype] => PLAIN
               [ifdescription] => 0
               [ifid] => 0
               [lines] => 1
               [bytes] => 25
               [ifdisposition] => 0
               [ifdparameters] => 0
               [ifparameters] => 1
               [parameters] => Array
                  (
                     [0] => stdClass Object
                        (
                           [attribute] => charset
                           [value] => UTF-8
                        )

                  )

            )

         [1] => stdClass Object
            (
               [type] => 0
               [encoding] => 0
               [ifsubtype] => 1
               [subtype] => HTML
               [ifdescription] => 0
               [ifid] => 0
               [lines] => 1
               [bytes] => 103
```

```
            [ifdisposition] => 0
            [ifdparameters] => 0
            [ifparameters] => 1
            [parameters] => Array
               (
                  [0] => stdClass Object
                     (
                        [attribute] => charset
                        [value] => UTF-8
                     )

               )

         )

   )
)
```

The above structure communicates to the email client if the body is in plain text, if there is an mime attachment, if there is an inline image and if this is a response to a previous email. The function _get_body_attch() will parse the body of a message, sort the pieces and prepare the body for display.

Our function will return the body and/or attachment in the message without mime characters, header information or quoted printables. The function is contained in the functions.php file.

Next we consider the implementation of sending mail messages. To send an email, we first determine if it's a reply to a previous email or a new email. If it is a new mail, the user fills out the form fields and the form is forwarded to status.php where the input fields are verified on the server side. If the user entered valid email and an optional name, a new mail object is created for the PHPMailer class. We use PHPMailer instead of the naïve mailer class. PHPMailer is an open source library that broadens the native mail library's capabilities. Most of all, PHPMailer supports authentication at log in while the native mailer does not have the option of signing in securely to an email server.

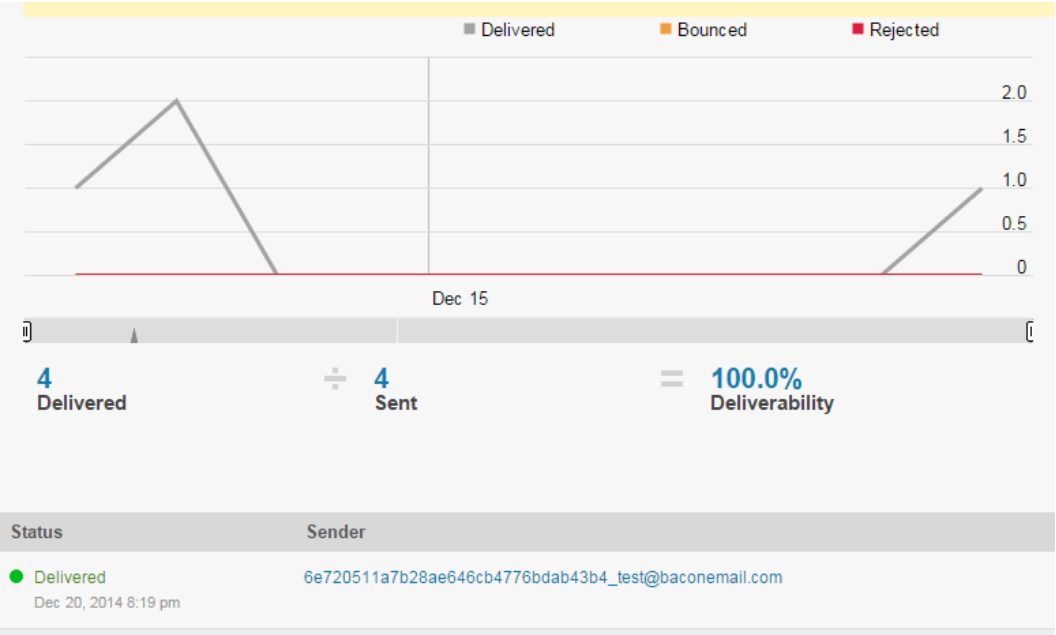
All the available information is passed on to the mail object, including the “from email” field. At this point, we check against the database if such an email address already exists, if not we add it to our contacts for future reference. Finally, we call the send() method within the mail class. Upon success, we alert the user and forward them to the inbox.

To improve performance and ensure deliverability of the emails, we have implemented outgoing email to be sent through mandrillapp.com. There are several advantages to that; first mandrillapp.com has already established reputation among email servers so email is not delayed as much. Greylisting is the process where email servers reject all first time email messages from an unfamiliar IP address. As most mail servers that distribute spam do not use standard MTAs that follow RFC compliant protocols. Intact, more often than not, such servers do not retry sending the email. If a spammer were to retry, the greylisting delay may provide enough time for the IP to be blacklisted if enough statistical information is gathered about the IP’s activities. (Silicon.dk Aps, n.d.)

If we send email from our mail server, users will experience unusually long delays while ISP providers place our IP address on a white list.

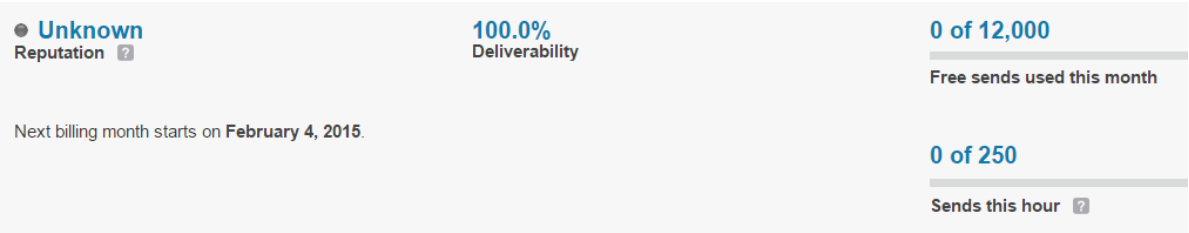
Another benefit of using Mandrill as our outgoing server, is added functionality of deliverability tracking, average open rate and average click rate statistics.

Figure 15 Mandrill features and interface.



Most importantly, we find the feature of reputation essential to our implementation. One of the areas of concern when implementing a private email system is that it can be misused by spammers. We do not want to propagate the behavior we are trying to fight. Mandrill allows us to create subaccounts, where we can monitor the reputation of an account and obtain real-time statistics with each rejected email, or about each email reported as spam. Mandrill offers an extensive API for PHP and other popular web languages with RESTful services. Furthermore, Mandrill automatically limits the quantity of emails that can be sent per hour based on a user's reputation and alerts us via email of the accounts performance. These are all must-have features in a unique channel email system to avoid abuse.

Figure 16 Reputaion and Deliverability score.



TESTING AND RESULTS

To test the effectiveness of the email system, we obtained a fully qualified domain name from Godaddy.com and modified the zone file's mx records to point to our Amazon EC2 RHEL email server. We also acquired an elastic IP address and ensured that it is not blacklisted on mxtoolbox.com. Mxtoolbox varieties against 300 known blacklists such as Spamhaus DBL.

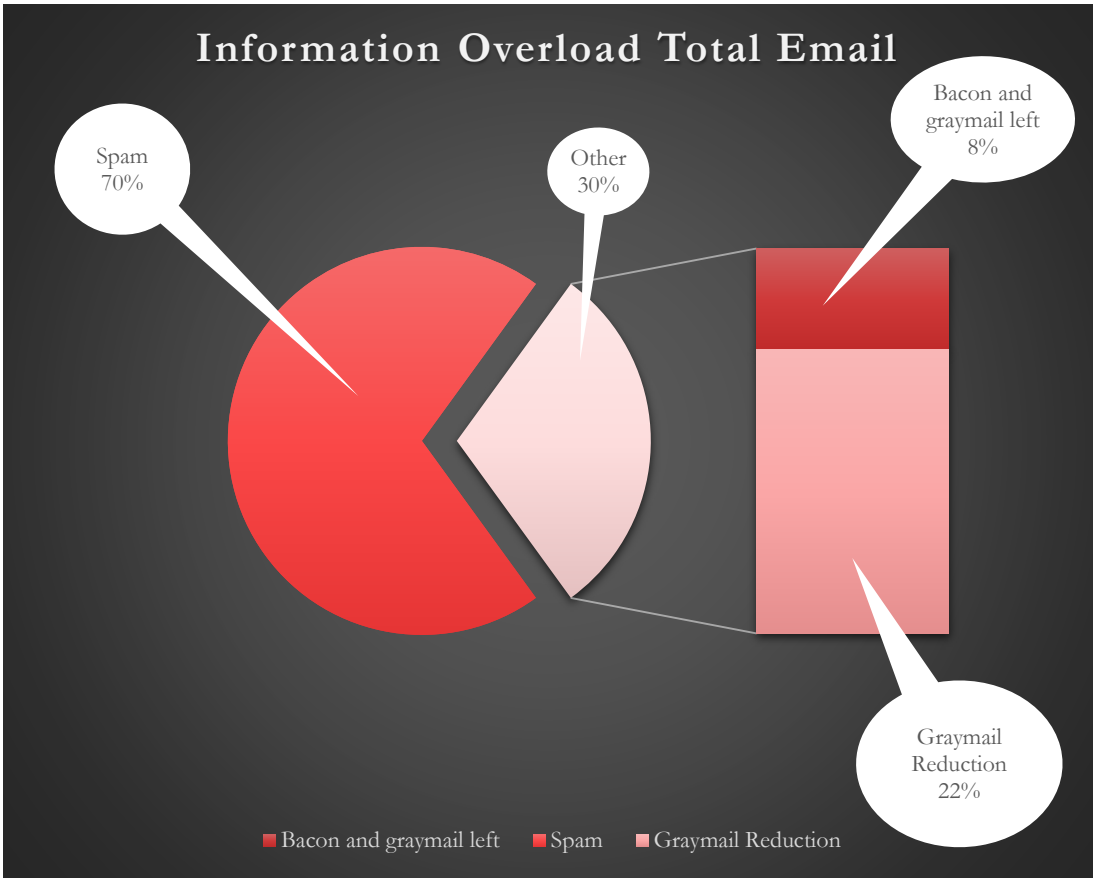
The first domain name contained, a new top level domain, namely ".email". This proved to be a problem with some websites as it was rejected as an invalid email. We also started out with the following format: uniqueid.test@domain but found that a few websites also considered the dot in front of the "@" symbol to be an invalid address. Eventually, we acquired another domain name with a .com top-level domain and settled on the underscore instead of a dot between the unique id and the username. This combination appears to be acceptable for all websites that have requested a valid email address. Since November 21 we have assigned over fifty unique email addresses to various contacts. Surprisingly, none of the websites divulged our email address to third party and it does not appear to be a common practice. However, our email box quickly started to fill up with newsletters and unwanted emails. Fortunately, our system allows us to unsubscribe from all of the newsletter in less than couple of minutes.

For better understanding of the pattern in an inbox, we created a custom script that parses a typical Gmail inbox. We analyzed all email received in the inbox for the year 2014. There were 1358 unique senders. The total emails received in the inbox were 9452, of those 2485 were opened, resulting in 26.29% open rate. After sorting the results by open rate and deleting all email under 65% that were NOT opened (6887), we accomplished a 72.86% reduction in information overload for non-spam emails. Furthermore, if we were only to delete the emails

with 0% open rate, there would be a 19% reduction. By examining the emails of importance, we see that they possess at least 85% open rate, but most often they exhibit 100% open rate. Therefore, the open rate threshold of 65% can be safely increased depending on the user preferences. It can even be increased to a 100% because that simply means, if a user does not open an email, it is eventually deleted.

If consider that our system can eliminate close to 100% spam, the total savings, considering an additional 70% spam are approximately 92%. This translates to large quantities of storage space and productivity savings.

Figure 17 Results



CONCLUSION

The price per gigabyte of physical storage has drastically decreased from $437,500 to about $0.05 but it doesn't mean that we can be wasteful and store more than necessary. In 2011, Google revealed that its data centers consume more than 260 million watts at any point in time, enough to power 200 000 homes.(5) In 2010, the company used 2,259,998 MWh, contributing 1.46 million metric tons of carbon dioxide. (Business Sector Media LLC, 2011) Storing unread mail is wasteful.

In conclusion, we have identified the problem of information overload in email systems that has not received considerable attention. Current practices, force users to divulge their private emails, in exchange for accessing content on a website and getting bombarded with unwanted emails. Email providers have done an excellent job of keeping spam out of user's inboxes but have not been as effective at empowering users to control the amount of graymail which accounts for approximately 80% of all non-spam email.

Users are left with two options; one setting up custom filters or two, or perusing an unsubscribe link by the third party that send the email. The first option can be beyond the average user's technical abilities and certainly can be time consuming. One has to identify overtime, which emails they are not interested in and constantly have to evaluate who the sender is and how to design a filter that will prevent all future emails from the sender. After all that effort, a clever and desperate advertiser may simply change their email address and bypass the rules with ease. Option two is no less promising. First, the user has to rely on the sender to include an unsubscribe method. Next, advertisers don't always like to see their readers go so they often make those wishing to unsubscribe fill out a survey, asking questions such as if they would

rather change contact preferences in their account or limit the amount of email they would receive or even sign in. This process already has taken the user's attention away from more productive activities.

Even worse, some advertisers, use attempts to unsubscribe from their service as a way to confirm that the email is actually genuine, working email address that is monitored and checked by an active user. This could result in even more spam and graymail.

We have proposed a solution that gives users a full control of their inbox. A key feature is the ability to know where email arrives from. We accomplish that by aliases that are assigned to each contact while not having a receive-all email. Currently users have several email accounts that they prefer to give out in different circumstances, this system of unique email aliases accomplish that without having to manage multiple accounts. By assigning an alias to each contact, we gain the ability to monitor incoming email on per channel basis rather than on individual basis, eliminating the need for expensive and performance intensive spam detection and priority classification. In addition, we can easily control the amount of information flowing in through a channel with simple one-click unsubscribe.

We have learned that open rate is a great way to determine if an email is graymail or quality email. Users can choose to let our email system function on auto-pilot and let it manage information inflow based on open rate history for each channel. We have determined that incoming email with less than 65% open rate can be assumed to be graymail. Although graymail is not of highest priority, users may not want to entirely unsubscribe from a service because on rare occasion, they see something of interest in a newsletter or promotion. Such is the case with Groupon or social media.

Our system, upon detecting a low open rate, can trigger an auto delete function that will give a user as much time as necessary to open such emails within a calculated time frame. Some users may only check email once a week so it would not be sensible to delete their low-open-rate emails within three days from receipt. Users need to be given a chance to open an email. Once an email is opened, it will not be deleted while the unopened email messages are deleted after the estimated time frame. There is also the added convenience of color transition visual, to warn the user when an email is approaching deletion. This may even encourage a more prompt response pattern.

Furthermore, our unique channel email system, when used in auto-pilot, will unsubscribe users from incoming emails that fall below 10% open rate within 6 a month period, while also deleting all unopened emails from such channel. We have determine that email with less than 10% open rate is actually of no interest to the user anymore and that users would really rather not receive such email.

The above described solution accomplishes a 100% spam detection and reduces information overload by 92%, simply by allowing users to control the content they receive. Users can enjoy three modes; A) Auto-pilot, where the system gives no warning, but takes cautious steps to manage low open rate email, even unsubscribing when necessary. B) Warning mode, the email system performs the operations that auto-pilot would but issues warnings to the user before taking action) Manual mode, allows the user to manage the email box completely on their own, by providing them options that they can use to easily manage information inflow. Such features include one-click unsubscribe and various timers that can be set for each channel.

Additionally, to avoid malicious use of our email system, we are using a third party service as our outgoing mail server. Mandrill offers, reliable email delivery with a host of useful features such as delivery tracking and most notably reputation monitoring based on spam complaints. The service is equipped with elaborate RESTful API that can disable an account in case of spam activity. It also limits the amount of emails that can be sent at any given hour based on reputation.

Finally, to ensure that more users can take advantage of clutter-free email, it is important to consider how our email system can be integrated with existing email accounts such as Gmail and Outlook.

There are several approaches to integrate with third party web clients; A man-in –the-middle approach would entail using the custom email system described in this paper as a relay engine. The user formats the outgoing email as to encapsulate the final destination but send it to the custom email server. For example, if a user is attempting to contact johndoe@gmail.com, the "to" field would appear as 'johndoe@gmail.com'_user@domain.com. Once, the custom email server receives an email in this format, it automatically forwards the email to johndoe@gmail.com, masking the true sender's identity and creating a new alias. If the user wanted to reply to the email, it would simply arrive at the custom email server and forward on to the user's private email. The down side of this approach is that, our system has no control over how the contacts are organized in the third party email and may result in confusion. What's more it will be of no help with other incoming email.

Alternatively, the user can log into our custom email server with their third party credentials. In this way the interface can mimic that of the third party but the functionality is that

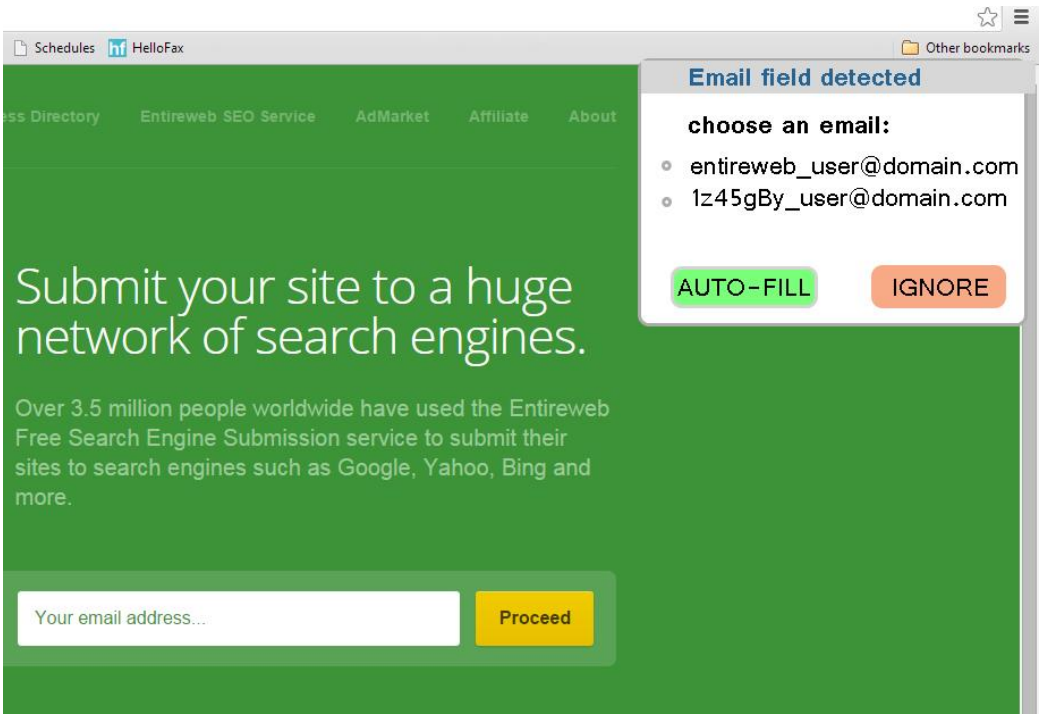
of the described email system. This approach gives the email system most control over managing the third party email account and can even synchronize contents on both servers by using the IMAP protocol. A possible challenge is to build up enough reputation to convince current third party email users such as those with Gmail accounts to divulge their log in credentials to a provider other than the one offering them the email account. Additionally, it is difficult to synchronize current contacts and issuing new custom email channels. It is essentially, too late to hide an email address that has been used for previous correspondence.

Another common approach used to integrate different services together is the web browser extension. In this scenario the user logs in to their third party web client but a browser extension serves as a negotiator. It would mostly be used to keep track of the address book required to manage a unique channel email system and would interfere minimally with the way the user is already using their email account. The extension does not need any sensitive information form the user that may make them feel uncomfortable about sharing. They simply need to provide their private email address. The extension manages the address book in the background but forwards the email to the email address that is stored. Although this approach is straightforward, browser extensions have failed to gain momentum and would not give the user a complete control of their email. Anti-virus software tend to block browser extensions and can cause compatibility issues.

The most consistent results can be accomplished by switching completely over to the unique channel email servers and starting a new our email system.

In implementing this concept, we chose to build a custom web client where a user can add a contact manually. Creating contacts before communication starts, places the burden to stay disciplined in managing email before it arrives in the inbox. That is why the process of adding a contact must be as efficient as possible. We believe that a browser extension will go a long ways to relief the user from going back to the web mail client to add a contact. A simple browser extension can detect when an autofill form field is present then a pop out with a prompt. The prompt could offer the user to generate the email, fill it out for them and save it to the database, without ever having to leave the website.

Figure 18 Autofil prompt upon email request detection.



Additionally, let's consider all the scenarios where a user can be asked to provide a valid email address. By far, the most practical application of our email system is when a user is

prompted for an email while visiting a website. Our current implementation can handle this scenario fairly well and can do better with the aid of a browser extension.

Scenario two, we publish the email on a website, forum, resume, document, letter-head, business card, poster, advertisement, cc field of a mass email and cetera. Our email system cannot offer a one-to-one performance. We can still create a different channel for each of the forms mentioned above with the benefit of being able to track where contacts found our email but the limitations are obvious.

Scenario three, we exchange the email with someone verbally in person or over the phone, or write it down for them while interacting. If we don't know the individual extensively, we will need to provide them with a new email on the spot. It would be convenient, if we developed an app for a mobile device to accompany the email system in this situation. The app would allow us to enter into the system the email we just invented shortly after providing it to the other party. Additionally, to avoid stressful situation of coming up with a valid email on the spot, and to avoid overlap with another contact, the app can have a simple email already generated that we can simply read off the screen. Something along the lines of a username followed by a two to four digit number and the domain as depicted in the next figure.

Figure 19 Simple app to streamline in-person exchange.

Finally, our current version does not provide support for multiple recipients, although adding support for such a feature would not require extensive modification to the code. The user can enter all desired contacts in the “to” field and then the mail function would be called in a loop, while matching the “from” address with the “to” address of each contact by querying it from the database.

Reducing email overload is an exciting challenge that is yet to be addressed. It was not until the last several years that email has become so widely adapted. Email has benefited tremendously as the war on spam has seen many victories. The next great challenge is to place

the user back in control of their reading preferences, and not be forced to endure a flood of unwanted messages.

REFERENCES

Avodele T. Shikun Zhou, K. R. (2009). intelligent email prediciton system (IEPS). *Internet Technology and Secured Transactions* (pp. 1,5). ICITST INternational Conference.

Douglas Aberdeen, O. P. (2009). *The Learning Behind Gmail Priority Indox*. Retrieved from Research.google.com: http://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/36955.pdf

Fu, X. (2014). Hwo to Send a Self-Destructing Email: A Method of Self-Destructing Email System. *IEEE International Congress* (pp. 304,309). BigData Congress.

Golden, B. (2013). Amazon Web Services For Dummies. In B. Golden, *Amazon Web Services For Dummies* (p. 384). Wiley.

Google Inc. (n.d.). *Using an adderss alias*. Retrieved from support.google.com: https://support.google.com/mail/answer/12096?hl=en

Google Inc. (n.d.). *Block Unwanted Emails*. Retrieved from https://support.google.com/mail/answer/8151?hl=en

Guy, L. (n.d.). *How to make sure your emails land in Gmail's priority inbox*. Retrieved from convinceandconvert.com: http://www.convinceandconvert.com/email/how-to-make-sure-your-emails-land-in-gmails-primary-inbox/

Heinlein, P. (2014). In P. Heinlein, *Dovecot: POP3/IMAP servers for enterprises and ISPs* (p. 376). Open Source Press.

Kim, T. J. (2003). A downemail syste: an internet email system based on sender mailbox. *The 9th Asia-Pacific Conference* (pp. 87-90). APCC.

Kyle Dent D., W. V. (2013). Postfix: The Definitive Guide. In K. D. D., *Postfix: The Definitive Guide* (p. 280). O;Reilly Media, Incorporated.

Microsoft. (n.d.). *Use aliases to add email addresses to your account*. Retrieved from http://windows.microsoft.com/: http://windows.microsoft.com/en-us/windows/outlook/add-alias-account

Qiong ren, Y. M. (2007). SEFAP: An Email System for Anti-Phishing. *Computer and Information Science* (pp. 782,787). ACIS International Conference.

Sara Radicati, J. L. (2013). *Email Statistics Report, 2013-2017*. PALO ALTO: THE RADICATI GROUP, INC.

Silicon.dk Aps. (n.d.). *gerylisting.org*. Retrieved from http://www.greylisting.org/

Spira, J. B. (2009). *Intel's War on Information Overload: A Case Study*. New York: Basex, Inc.

McLaughlin, Brett. PHP & MySQL: The Missing Manual. Sebastopol, CA: O'Reilly Media, 2012. Print.

Turnbull, J., & Lieverdink, P. (2009). Pro Linux system administration. Berkeley, CA: Apress ;.

Ng, B. H.; Crowell, A. & Prakash, A. (2012), Adaptive semi-private email aliases., in Heung Youl Youm & Yoojae Won, ed., 'ASIACCS' , ACM, , pp. 69-70