Tipsy
by Philippe Schiff

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology

June 2015

The author hereby grants to M.I.T. permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole and in part in any medium now known or hereafter created.

Author: ____________________________
Department of Electrical Engineering and Computer Science
June 5, 2015

Certified by: ____________________________
Department of Electrical Engineering and Computer Science
June 5, 2015

Accepted by: ____________________________
Prof. Albert R. Meyer, Chairman, Masters of Engineering Thesis Committee
In this document I describe Tipsy. Today’s models for paying online content creators are inadequate. With Tipsy, the idea of Voluntary Microdonations is revisited and modified to overcome the obstacles of previous and current attempts. I implemented a Google Chrome browser extension that logs a user’s content consumption online and stores the information locally. Tipsy does not require any user input to decide which content is of particular interest to the user. Instead, Tipsy suggests a payment based on the user’s browsing time. Tipsy reminds the user to pay and suggests a dollar amount. The user can pay the content providers directly. Tipsy uses no server, aside for some completely anonymous data gathering, and does not collect any money.
1 Introduction

Today’s models for paying online content creators are inadequate. With Tipsy, we revisit the idea of Voluntary Microdonations and modify it to overcome the problems with attempts made so far. I will describe my implementation of a Google Chrome browser extension that logs a user’s online content consumption and helps the user pay for the online content they consume. Tipsy does not require any user input to decide which content is of particular interest. Instead, Tipsy monitors the user’s browsing activity and suggests a payment proportional to the user’s browsing time. Tipsy acts like a subscription to the internet. It reminds the user to pay and tells the user how much they owe after which they can pay the content providers, as in Figure 1. The user pays the content providers directly. Tipsy utilizes no server and does not collect any money.

In the following subsections I will elaborate on the reasons for Tipsy’s development. I will identify the problem and then describe the various attempts made at solving it.

1.1 Paying for Online Content Creation

In traditional commerce, consumers pay producers directly. Producers generally accept money in return for handing over goods to consumers. The cost for producing the goods can
Online content creation has costs as well. The creation itself costs time and money. Other costs are often server side hosting and maintenance.

However, in online or virtual goods, the model is different. Consumers of online content do not always pay for the content they consume the way consumers of more tangible goods do. This has led to a number of different approaches to generate revenue from online content.

In the next three sections I will describe three main mechanisms that have been used to pay online content providers.

1.1.1 Paying for Content Directly

One method has been to require consumers to pay directly. Netflix charges for videos and some news websites have various forms of a paywall set up. This method has been adopted in some industries. It has worked well in the entertainment and movie industry (think of iTunes or Netflix) but has had limited success in the media industry [1].

1.1.2 Ads

The second and most widespread method for paying for online content is advertisements. Content creators are paid by companies to advertise their goods or services on the content creators’ website. This allows the creators to receive payment for their creation costs. This method has several significant drawbacks. First, advertisements frequently detract from the content provided. The amount of ads on a simple Google search is arguably an example of this. Second, by not having money flow directly from the consumer to the producer, the obligation to pay is felt less acutely. This has given rise to tools like Adblock [2] which rob the content creator of payment. In addition, advertisers might try to prevent content providers from including content which the advertisers don’t approve of. Such conflicts of
interest can create a form of almost imperceptible bias or even censorship.

Another argument against using ads to fund content creation is that a surprisingly large number of companies that rely on it have yet to post a profit. Twitter, tumblr, yelp, Pinterest, Instagram and Spotify, are all examples of this flaw [3]. Ads fail to adequately remunerate content creators. The news media industry in particular is in need of a new payment model. As Figure 2 shows, the newspaper industry’s rise in revenue from online ads is less than the fall in print ads. It clearly is not a perfect system.

An additional drawback of the ad model is that due to the competitiveness of the ad market, many services track users to learn how to serve even more effective ads. Content providers (or search engines) will place cookies in the users’ browser or use other methods to learn about the user. While this is often unknown to users, awareness of privacy concerns is starting to spread, giving rise to tools like the Don’t Track Me extension [4].

Last but not least, ads are very annoying.

1.1.3 Pay What You Want

Another solution for paying for online content is the Pay What You Want method (PWYW). This method has been used before. Radiohead famously released their “in Rainbows” album
online with a PWYW payment method. While they did not release their earnings from the album, it was found[6] that in the U.S. 40% of downloaders paid an average of $8.05 (36% and $4.64 respectively outside the U.S.) which indicates that people do pay voluntarily even when they don’t have to. Wikipedia, which periodically runs donation campaigns, received $20 million in donations in 2012 [5].

In a new book, “The Art of Asking”, Amanda Palmer documents [7] how she was able to make a “steady and predictable income every single day” from giving free street performances. She gives a number of criteria that need to be met to generate this steady income (e.g. authenticity, open and clear exchange).

Humble Bundle [8] allows users to pay whatever they want for some content (oftentimes movies or computer games). They have so far processed nearly $1 million in total payments from almost 160,000 users [8].

There are a number of services like Flattr [9] and Gratipay [10] (see section 2) that make it easier for users to reward online content voluntarily by either liking some content or actively selecting to donate to a particular content creator. We see that the PWYW scheme, though not yet widespread, is in some cases workable and is already in use on the internet.

1.2 PWYW for All Content

This raises the obvious question: would PWYW work for all internet content? Would people pay voluntarily for content they consume on the internet? One of the reasons they might not is laziness. Many users may feel too inconvenienced to have to pay attention every time they consume internet content. Other issues with voluntary donations are transaction costs for micropayments (e.g. PayPal’s $0.30 minimum), privacy (users don’t want all their browsing data logged), trust (users need to trust a payment service) and forgetfulness (a hypothetical non-lazy user could still forget to pay).
Another issue that might prevent users from voluntarily paying for online content is that even if they decide at a given point to pay, they don’t necessarily know how much they would want to pay since they do not know how much their current decision would fit into their donation budget. One could imagine a user wanting to pay $x$ amount of money per $t$ amount of time, but they might not know how much their current browsing fits within their total browsing, especially since future browsing would have to be taken into account. A successful solution would have to allow users to specify a fixed amount of money they want to pay per time $t$ in bulk and divide the total based on usage.

Additionally, in order for a potential solution to work optimally it would have to become pervasive and trustworthy enough to minimize costs and encourage adoption in order to eventually become a standard for subscribing to and paying “the internet”.

As was described by Amanda Palmer [7] people need to feel that their voluntary donations are conducted in an open manner such that it is clear how the author is being remunerated. Therefore any solution would have to ensure that the method by which content creators are paid remains unambiguous. By using existing payment processors (such as PayPal or Dwolla) and handling payments directly from their sites user trust could be most easily maintained.

Therefore, a solution for paying for online content would ideally have the following features:

1. Allow user to pay creator directly
2. Minimize or abolish transaction costs
3. Allow for division of bulk payments
4. Require minimal effort on user’s side
5. Pay content providers fairly
6. Follow a subscription model

7. Respect user privacy
2 Previous Work

Micropayments have been tried before, and there are still a number of applications offering this service today. In this section I describe them and the reasons why they are not very widely adopted.

2.1 Flattr

In Flattr [9] a user can add money to their account and decide on a budget in advance. Whenever a user comes across content they like they must “Flattr” it (by liking, upvoting, etc.). Flattr divides the total budget by the amount of Flattrrs and sends the money to the creators. There is a $2.00 minimum and money is distributed monthly. While Flattr does allow users to choose a budget, Flattr’s main drawback is that it requires the user to explicitly Flattr content. There is no way to automatically Flattr content. This requires some effort - especially for users who don’t necessarily like or upvote content that they like. In addition, Flattr takes a 10% cut of all payments. Flattr is thus relatively fair but has minimum costs, takes a cut, and knows all the user’s upvotes and likes. While innovative, Flattr does not have all the features listed above.

2.2 Gratipay

Another micropayment platform is Gratipay (formerly Gittip) [10]. In Gratipay the user designates the content providers they want to pay and the amount they want to pay. Gratipay aggregates the funds before distributing them. Money is distributed weekly and there is a very low minimum of $0.01. While Gratipay is arguably fair, the payment amounts are not proportional to the amount of time the user spent consuming the content. It also does not have all the other features mentioned above since it collects user information and does not allow for a fixed budget selection.
Both Gratipay and Flattr have the additional drawback that they follow a model where users only pay for the content they like instead of fostering the idea that users subscribe and pay for their entire browsing activity.

2.3 Contributor by Google

Google has recently announced its own proposed solution: Contributor by Google [13]. Google calls it “an experiment in additional ways to fund the web”, and it seems to work as follows. The user signs up and chooses a monthly contribution between $2 and $10. Then, whenever they visit a site a “pixel pattern” or a “thank you message” will appear where the user might usually see an ad. The contribution gets used to pay the site who would have hosted the ad.

Visit participating sites.

When you visit a participating website, part of your contribution goes to the creators of that site. As a reminder of your support, you’ll see a thank you message - often accompanied by a pixel pattern - where you might normally see an ad.

Figure 3: All Google says is that “part of” the contribution will go to the site.

There are a number of drawbacks to Google’s Contributor. The first and main one is that Contributor only allows for payment to sites that show Google ads and thus are part of Google’s advertisement network. Sites that are unable or do not wish to use Google’s ad system will not get any of the users’ contribution. Sites that do not want to display ads at all will also not make any money from Contributor. Contributor seems to be built on the assumption that sites will show you an ad, but if you don’t want to see it you can pay. It does not leave for the possibility of simply supporting content directly for sites that don’t have ads in their business model. Content providers that don’t display ads in the first place (e.g. Wikipedia) will be unable to generate any revenue from Contributor.

Additionally, it is unclear how Google will divide the contribution the user sets. Figure 3 shows a quote from the Contributor page and the information the user gets about the
distribution of funds. Google is (hopefully) not going to track the time the user spent on each site and will thus not have a way to remunerate providers based on how much the user valued the content. If they do, that would bring back the problem of having the user's detailed browsing activity stored outside the browser. It also doesn't appear that the user has any control over who gets paid or the amount that is paid. Finally, Google Contributor still involves payment through Google instead of to the provider directly. Google will presumably take a cut, thereby reducing the amount that the creator receives.
3 Features

In this section I will describe Tipsy’s features, the design trade-offs that were encountered and the choices that were made.

The first subsection will give an overview of Tipsy’s features, and the rest will elaborate on the specific details of each of them.

3.1 Overview

I developed Tipsy under the supervision of Professor David Karger and in collaboration with Tim Branyen [11] from bocoup [12]. Tipsy is a browser extension that installs in a user’s browser. I developed it for the Google Chrome internet browser. The user chooses a payment rate, and Tipsy then logs the user’s site visit information. Since it is a browser extension all information is logged within the browser only. When a user is ready to pay, Tipsy displays the payment amount it generates and initiates a payment session upon user request. Tipsy does not collect any money and is server free. Although payment amounts and mechanism are suggested by Tipsy, the payments are completely voluntary. Tipsy does, however, try to encourage users to pay by using more imperative terminology (e.g. “Pay” vs. “Donate”).

3.2 Calculating the Ideal Payment

In this section I will describe how Tipsy accumulates charges in order to suggest to the user how much the content creators should be paid. In the charge accumulation, Tipsy keeps track of what the ideal payment should be according to a user-chosen metric. Tipsy provides the user with a view of how much should be payed.
There are two ways to accumulate charges, either with (1) a Calendar Rate or (2) an Activity Rate.

### 3.2.1 Calendar Rate

A Calendar Rate accumulation charge allows the user to specify how much they are willing to pay “the web” per some defined time. This way the user can know exactly how much they will have to pay come donation time - which makes the Calendar Rate Tipsy’s preferred payment rate. While a Calendar Rate gives the user a good idea of how much they will have to pay, if the user chooses a short time period (per day vs. per month) payments could become unequal if the user’s browsing pattern differs significantly between time periods. However, over long enough time, these variations should smooth out.

In addition, until many content providers display payment information in their source (see section 5.5), final payments suggested by calendar rates might seem inaccurate since sites that display this information will be overrepresented. For example, if there are only two sites that Tipsy can pay and the user chooses a rate of $10 per month, both sites would receive $5 at the end of the month even though it might not have been the user’s intention to pay that much to only two sites. Had more of the sites been payable by Tipsy then the division might seem more appropriate to the user.

### 3.2.2 Activity Rate

For users who want a more exact measurement Tipsy allows them to choose an Activity Rate charge accumulation so that they can decide how much they want to pay content providers per some defined time spent browsing the content (e.g. per minute, per week, etc.). Some users might like to base their donations entirely on how much time they spent consuming a given author’s content. While using an Activity Rate might obscure from the inexperienced user how much they would have to pay when their time to donate arrives, it
has the advantage that it is not affected by variations in a user’s browsing pattern. If a user spent a minute reading a blog on xyz.com on a given day but spent an hour the next, this will be accurately reflected in the final amount. To try to make it clearer to the user Tipsy shows an estimate, based on past browsing activity, of how much will likely be suggested for a fixed amount of time.

3.3 Donation Reminders

Tipsy generates reminders to make sure users remember to pay. In this section different ways to remind the user to pay will be discussed. There are three different methods from which Tipsy allows the user to choose.

The goal of the following reminders is to make sure that the user is reminded to pay at the right time. If the user is reminded when the bill generated might not yet be sufficient to warrant payment (especially if there are transaction costs, see section 3.5) then the reminder came to early. On the other hand, if the user is reminded when a large payment amount has already been accumulated then the user might get frightened and not pay at all (and even uninstall Tipsy). In that case, the reminder came too late. Therefore the defaults should be set so that the reminders trigger at the right time for reasonable use.

3.3.1 Fixed Period

This method reminds the user to pay at a fixed period (every day, week, month, etc.) regardless of the amount reflected in their donations page. It is the simplest setting but could be less useful for certain browsing profiles and payment settings. If after a month the user doesn’t have many authors to pay, the reminder would be bothersome to them. If after a month the user has a high bill, the reminder should arguably have come earlier.
3.3.2 Global Threshold

In order to make sure that the payment reminder comes when there is an appropriate amount to pay, a Global Threshold reminder could be used where the user is reminded to pay when their total bill amounts to a specified threshold. Of course, the usefulness of this method depends on both the user’s payment settings and threshold amount. If the user has a low rate which will not rapidly generate a large payment and a high threshold this reminder might not trigger often enough. Infrequent reminders could result in user surprise when a reminder comes with a large bill. However, with reasonable parameters the Global Threshold could be a good choice.

3.3.3 Local Threshold

This reminder is similar to the Global Threshold but instead of waiting to remind the user for the total browsing history’s bill to reach a certain amount, the user is reminded each time any of the individual content creators’ payment reaches a specified threshold. Figure 4 shows an example of this reminder. This way if the user wished to pay one payee at a time they can select to be reminded when the payee reaches the threshold amount. The Local Threshold reminder could be particularly useful with an Activity Rate because it is then natural to think of accumulating charges as the user visits the particular site.

Figure 4: A local threshold reminder.

3.4 Time to Pay

Whenever the user decides, whether they were reminded or not, they can choose to pay the different content creators for whom Tipsy has accumulated charges. These payment amounts
are calculated using the payment rate (see section 3.2) chosen by the user. Users can pay the individual content creator directly using one of the processors the creator accepts (see section 5.5). The user will thus be redirected to the payment site and on completion will return to Tipsy. Tipsy will register the amount that was paid and the entry will be removed until future charges have accumulated.

3.5 Issues With Individual Payments

While paying content creators directly does reinforce user trust and make the flow of remuneration clearer to the user, it also gives rise to what might be Tipsy’s greatest drawback: each payment must be made individually. There are a number of issues with requiring each payment to made individually. First, the user will have to go through as many payment sessions as there are payees, which could become cumbersome. A user would probably prefer to just click “Pay” and have all the payments process at once. Another drawback of individual payments is that they make it harder to solve the transaction cost problem. As mentioned in section 1.2, one of the the key elements any solution needs to have is that it must abolish or at least minimize transaction costs as much as possible. Making \( n \) individual payments through PayPal, which takes a $0.30 minimum charge, would results in a net \( n \times 0.30 \) dollars gone to PayPal when if all payments were made in a single session it would just be $0.30. This problem presents itself very clearly for payments that don’t go much above $0.30. These will get entirely eaten by the transaction costs. If the user has 20 payments each of $0.30, there will be $6.00 being owed that cannot be paid. See section 10.7 for a discussion on how to potentially solve this problem.
3.6 Logging Information

In order to calculate payments, Tipsy needs to log the user’s browsing information (see section 5.1 how this is done). Strictly speaking, Tipsy does not need to display this information to the user but it was found that some users enjoy having the ability to quickly scan all the sites that Tipsy has logged for the user whether or not a charge has been accumulated or if Tipsy can even pay them (see section 5.5). Therefore Tipsy gives the user this information.

3.7 Getting Tipsy

Tipsy has a landing page tipsy.csail.mit.edu where users and content creators can go to learn more about Tipsy. Figure 5 shows the top of the landing page.

3.7.1 For Users

All a user needs to do is install Tipsy. The landing page has a download button which takes the user to the Google App Store from where the extension can be downloaded and installed. Once that is done the user can continue browsing as usual.

3.7.2 For Content Creators

In order to be paid by Tipsy users content creators need to include some information in their content which tells Tipsy how to pay them. The exact format of this information will be discussed in section 5.5.

Tipsy’s landing page contains instructions for content creators on how to include this information in their content’s source and provides forms so that content creators can easily generate the different formats used. Tipsy also provides a WordPress plugin which makes it
Figure 5: Tipsy’s landing page.
easy for content creators using the WordPress framework to include this information without having to write any code. The plugin is downloadable from the WordPress plugin site.

Tipsy also has the ability to incorporate hard-coded author information for content creators who provide it.

Content providers might find the ability to check whether a user visiting their site has Tipsy installed useful. Tipsy provides a small library that provides this ability. Information on how to use it is on the landing page as well.
4 Tipsy’s User Interface

In this section I will describe Tipsy’s User Interface.

Tipsy was built to be as unobtrusive as possible. After Tipsy is installed, it will only display a little icon near the address bar, like most other extensions. The user can hide this button if they wish. Figure 6 shows Tipsy’s icon near the address bar.

The user can click on the icon, and Tipsy’s UI will open in a new tab. The first time the user opens Tipsy they will be able to choose their settings through the “Getting Started” pages which I describe in the next section.

4.1 Getting Started

The first time Tipsy is opened by clicking on the icon the user will be walked through a simple set-up where they can set their rate. Figure 7 shows what the users will see as they step through the set-up process. The inputs and layout of the getting-started screens are the same as what the user will see if they go to change their settings later, which provides for internal consistency.

The user can decide to skip the settings and default settings will be chosen. Since the Calendar Rate follows the “subscription model” more faithfully it was chosen as the default option with a weekly time span. The default interval reminder is set to once a week to correspond to the default rate period.
Figure 7: The user will see these pages when they open Tipsy for the first time. They can set their settings this way.
4.2 Log Page

Figure 9 shows the Log page which displays to the user all the browsing activity that Tipsy has logged for the user. The entries can either have author information or have no author information. The ones that do have author information have a green background color to their row while the others have a red background. The user can also easily get an overview of only the pages that have author information by clicking on the “Hide entries without author information” button (which is selected by default). See Figure 9a.

The user can also choose to see the individual visits (see Figure 9b), which can be done by clicking on the rows. The rows light up and get a “pointer” mouse icon which both suggest to the user that the rows are clickable to ensure learnability.

In order to allow for user control the user can delete their history by clicking on the “Clear History” button. To ensure safety, the user will first have to click a confirmation dialog (see Figure 8) that will try to prevent them from deleting their history by mistake. A additional useful feature would be to have Tipsy detect when the user is deleting their browser history and ask whether they would like to delete their Tipsy history at the same time. This way the user would not be surprised if their “deleted” history shows up again.

Figure 8: Clear history confirmation.
Visit log

This page displays a history of all sites you have accessed. Click on a site to expand and view each visit.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Author</th>
<th>Visit count</th>
<th>Days with visit</th>
<th>Time spent</th>
<th>Last time visited</th>
</tr>
</thead>
<tbody>
<tr>
<td>brayen.com</td>
<td>Tim Brayen</td>
<td>3</td>
<td>1</td>
<td>3 seconds</td>
<td>12:51AM - Tue, May 12, 2015</td>
</tr>
</tbody>
</table>

(b) The Log page with all visited sites and the rows extended.

Figure 9: The Log Page that the user can view at any time.
4.3 Settings Page

Figure 10 shows a view of the Settings page. On this page the user can choose their preferred rate and reminders.

Rates
The user can choose between the **Calendar Rate** and the **Activity Rate** (see section 3.2). The user can select their rate by activating one of the radio buttons. The unselected radio box’s rate will turn gray to indicate to the user that it is not being used. The amount and time inputs of the rates will keep displaying the last setting (or default) so that the user can easily see what setting they would return to in case they choose it. The amount inputs automatically filter out any non-appropriate input to ensure both feedback and safety.

As opposed to a Calendar Rate, an Activity rate might make it harder for a user to know
how much they will owe after a given amount of time. To make it easier for the user Tipsy provides some feedback containing an estimate of what their bill might be after two week. Figure 11 shows this. It is only an estimate but is based on the user’s browsing history of the past two weeks projected into the future.

Other rates might also be envisioned. For example, a rate could be chosen such that payments are proportional to the total amount of visits a site received. Tipsy could potentially also calculate payments based on the number of days a visit was logged. However, we didn’t think that these were as reliable a metric of the preference a user has for content as either the Calendar or Activity rate, which both take time as the basic metric.

**Periodic Reminders**

The user can set a periodic reminder that will remind the user to pay every interval where the interval is chosen by the user. The user can choose to be reminded every \( n \times t \) where \( n \in \{1, 2, 3, 4\} \) and \( t \in \{\text{days, weeks, months}\} \). In addition, the user can also choose to be reminded every \( d \) at \( u \) where \( d \) is a day of the week and \( u \) is a time. Figure 12 shows the interface that allows the user to choose the periodic reminder.

For the reminders the user can select any of them with a checkbox. Here again, the reminders not set are grayed out but keep the last value in their inputs so that the user can see what the behavior would be if they enable the disabled settings.

The periodic reminders display a little text.
to give the user feedback about when the next periodic reminder will notify.

**Threshold Reminders**

The user can also choose to use a threshold reminder. The Global Threshold Reminder will trigger when the total payment owed by the user reaches the threshold $T_g$ set by the user. Tipsy will calculate the amount in the background based on the rate that is set and when it reaches $T_g$ a notification is triggered. The analogous behavior holds for the Local Threshold Reminder where a notification will trigger when a single payment reaches a local threshold $T_l$, also set by the user.

Figure 13 shows the interface that allows the user to choose a threshold reminder.

The Settings page was designed to have the least text possible while still maintaining learnability. User tests have indicated that it is quite easy for new users to figure out what each of the settings and inputs mean.

It could be argued that the reminders should be more coupled to the rates. For example, if the user chooses a Calendar Rate the Periodic Interval reminder could be automatically set with a reminder interval similar to the rate interval. If the user chooses an Activity Rate it might also be useful to automatically set a threshold rate so that when a particular sum is reached the user gets reminded. One user study indicated some confusion about the reminders (see section 8.2). It was not chosen this way to provide the user the full freedom in deciding how and when to be reminded. Future work might be done to have some automatic setting of the reminders based on the chosen rate and allow the user to modify them under an “advanced” section.
All of the reminders can be set independently of each other so that each threshold reminder and each periodic reminder can be selected or unselected. This way the reminders can be set to trigger at the right time (see section 3.3.3).

Currently, when an interval reminder is dismissed by the user it will only reappear after then another interval has passed. While a “snooze” feature is provided by Chrome Notifications, in the case where the user dismisses the notification it might be better if Tipsy would start reminding the user more frequently (eg. once a day) until the user pays, or at least visits the contributions page.

4.4 Contributions Page

The Contributions page is where users go when they want to pay. This is the page that opens first when the Tipsy icon is clicked. When a reminder triggers a notification and
the user clicks on the notification Tipsy will also open to this page. Figure 14 shows the Contributions page.

The payment processors that a content creator accepts will be shown by the buttons that are displayed in the table. The user can click on one of the buttons and will be redirected to the corresponding payment processor's site to complete the payment. For user feedback purposes, Tipsy will notify the user that they are being redirected to an outside domain (see Figure 15). The user can also change the amount they pay by modifying the value in the Recommendation textbox. When they then go to pay the value will display the new amount. Currently, adding an amount to a payment does not affect the total so the amounts charged for any other authors do not change. Another way of handling this case could be that the total remains the same so the payments to other authors would change. Tipsy could allow the user to choose between these two options by providing a “fix total” checkbox, similar to the “fix aspect ratio” option when resizing images in image editing software.

After a payment is complete the entry that was paid will disappear from the table. The user can also remove an entry from this table by clicking on the “X” next to the corresponding row. The user will have to click a confirmation dialog before this action is completed, similar to the one in the Log page.

When there are many content creators to pay the user might feel overwhelmed and not wish
to pay. Tipsy’s interest is that the user pay so Tipsy will only show a few rows with the highest accumulated charge. A button will appear that the user can click to show (and hide again) the hidden entries of the table.
5 Implementation

In this section I will describe how Tipsy is implemented. Since it is a browser extension, Tipsy is written in HTML/CSS/JavaScript.

5.1 Logging

The first ability Tipsy needs to have is to be able to log the user’s browsing activity. For each visit Tipsy stores (see section 5.4) a visit object that has the following fields:

- **accessTime**: An integer that represents in ms since the epoch (1/1/1970) the time this visit began.
- **author**: Who the author is (can be blank) and any payment processor it accepts.
- **timeSpent**: An integer that represents in ms how much time was spent on the page.
- **paid**: A boolean representing whether this visit has been paid.

The number of days on which the site was visited is also stored as **daysVisited** so that Tipsy can tell the user how many days they have visited a particular site. Figure 16 shows an entry in the log for a visit.

The information in each visit could overlap with other visit instances. For example, different authors can have multiple sites and there might be multiple authors per site. There might also be multiple authors per page (see section 10.6). There is also the url and domain information for a visit. All this information is stored in the visit.
and are filtered accordingly when the appropriate field is needed for a particular computation or display. For example, when the payments are calculated the visits are then filtered for author, where author is assumed to be a unique descriptor of the payee \(^1\).

This collection of data is called the log. While it may seem to grow indefinitely, any entry that has paid set to true can be safely deleted in the case where the log gets too big (see section 5.4 for more details).

Getting a correct number for the amount of time spent on a page turned out to be non-trivial to implement.

A browsing session that includes just one window and one tab is simple enough. But once a user has multiple windows each with multiple tabs, it gets more complicated. A user can close a window, switch tabs, close a tab, open a new tab, open a new window, navigate to another site within a tab, navigate to another site from the same domain within a tab, refresh the page, leave the computer, watch a movie, fall asleep, etc. Each of these cases presents another situation that has to be dealt with.

To help calculate all the different cases I defined a switch as a point where it is assumed the users attention is completely removed from the previous page and completely focused on the new one (which can be null). I narrowed it down to the following four switch cases:

1. **Focus moved to a new tab**: When the user's focus is moved to a new tab (usually by clicking) a switch occurs. When focus leaves the browser window, a switch from the last tab to null occurs. The will also occur when the browser is closed.

2. **Tab URL changes**: When a URL field in the address bar changes a switch occurs. This case is more straightforward since the old page disappears. A refresh is a special case of a URL change since the URL changes to a new URL which happens to be the same as the old one. Tipsy will behave as if no switch occurred.

\(^1\)Which might not necessarily be the case since one could conceivably have an author place payment information for someone else.
3. **A window is opened or closed:** When a window is opened or closed multiple tabs may appear or disappear. A *switch* thus happens for the focused tab that appears or disappears. Opening a new window (in some settings) may create a number of tabs but the focus will be on one only.

4. **Idle:** When the user goes in and out of idleness a *switch* occurs either from null to a URL or vice versa. Idleness is defined to occur when the browser has not detected any user action within a given time span \(^2\), the idle time. This case is needed for when the user just leaves the computer without exiting anything. The idle time, currently 20 s, is read from a set of defaults that Tipsy keeps.

At each *switch* Tipsy checks whether the new page is the same as the old one. If it is, nothing happens. If it is not, then (1) Tipsy subtracts the current time from the `accessTime` of this page to create the `timeSpent` and (2) Tipsy creates a new entry in the log for the new page and sets the current time to be the `accessTime`. When the new or old page is a null then (2) or (1), respectively, will not be done.

The above four cases can be listened to by the Chrome extension API \([14]\) using the following functions:

- `chrome.tabs.onActivated`
- `chrome.tabs.onUpdated`
- `chrome.tabs.onFocusChanged`
- `chrome.idle.onStateChanged`

With this, Tipsy is able to track how much time the user spends on each site. This information is used to calculate the payments and is displayed to the user.

These four *switch* cases and two-step read/writes take care of nearly all cases. There are some cases that Tipsy does not yet capture. One thing that can happen is that since the \(^2\)This means that the idleness could happen while watching a video, a limitation Tipsy currently has.
timeSpent is only updated when the tab is left, if the browser crashes an entire page visit could be lost. The danger of this could be mitigated by periodically updating the timeSpent while the user is on the given page.

More importantly, the switch model does not capture the case where a user consumes content from two tabs simultaneously. This could happen, for example, in the case where a user is listening to music playing on one tab and reading an article on another. One could argue that Tipsy is missing the attention the user is giving to the music content being played on the unfocused tab. The switch model currently assumes that the user’s focus can be either nowhere (null) or in a single place. The model can be extended to have switches to or from more than one place. We felt that this case is not that common, and given the absence of an easy way to detect that music is playing from another tab, \(^3\) it was not included in the current implementation.

5.2 Options

In this section I will describe the implementation of the user options. The user’s chosen options, or settings, are stored within the browser (see section 5.4) in settings.

5.2.1 Rates

As mentioned in section 3.2 Tipsy allows the user to select either an Activity Rate or a Calendar Rate. The rates are represented as donationGoal instances with type, amount and period attributes.

**Calendar Rate**

For the Calendar Rate of \(x\) dollars per \(t\) time interval Tipsy has the following object:

\(^3\)Recent versions of Chrome have started showing a little “speaker” icon on the tabs that are playing sound but as far as I was able to tell Chrome does not (yet) provide an API to tap into that information.
donationGoal = {
    type: 'Calendar',
    amount: x,
    period: t
}

The donationGoal that is currently selected is stored in the settings.

If Tipsy detects a donationGoal.type as 'Calendar', it divides donationGoal.amount proportionally between all the sites that are to be paid. The user’s payment intervals might not correspond exactly to donationGoal.period so Tipsy will correct for that by taking a fraction or multiple of the donationGoal.period as necessary.

**Activity Rate**

For the Activity Rate of x dollars per t time interval Tipsy has the following object:

donationGoal = {
    type: 'Activity',
    amount: x,
    period: t
}

If Tipsy detects a donationGoal.type as 'Activity', it finds the overall timeSpent for a given recipient and calculates the amount to be paid as:

\[
\frac{\text{timeSpent}}{\text{donationGoal.amount}} \times \text{donationGoal.period}
\]

When the Activity Rate is selected, Tipsy also gives the user an estimate of what the user’s payment will be after some time based on the user’s past browsing history as logged by
Tipsy. This is simply calculated by projecting past activity into the future.

5.2.2 Reminders

The reminders show a Chrome notification using the `chrome.notifications` API [15]. This API allows for triggering a Chrome notification immediately but in order to display the notification at a specific time, which is needed for the periodic reminders, a listener is set that waits for a Chrome alarm (from the `chrome.alarms` API [16]) set to go off at the interval chosen. When the alarm goes off the listener will display a notification and the alarm will be rescheduled for another interval. The reminder parameters and the information about which ones, if any, are enabled are also stored within `settings`.

5.3 Payment Buttons

Tipsy needs to be able to pay content creators directly. Ideally payments would happen completely on the payment processors’ side. This means that, for example, in the case of PayPal the user would be redirected to a PayPal page and see the payment amount field already populated with Tipsy’s suggested donation. This way the user does not have to trust Tipsy and can make the payments effortlessly.

Each payment processor handles their payments differently but most of them provide a button that can be placed on a page. The problem is that the button they provide is meant to handle payments from multiple users to a single entity (the page the button is placed on). In Tipsy we want to generate buttons for each author. In order to do this safely, we inject code into the client-side page that will generate a button for us using the information provided by the page that was visited.
**PayPal**

For PayPal this is simple. We can just place a button using PayPal’s format, and the only information we need from the content creator is their PayPal email address. With this we can create the button that will open a payment session on PayPal’s site and will redirect the user back to Tipsy when it’s done. The information about the success of the payment is returned to Tipsy.

**Dwolla**

We incorporated Dwolla into Tipsy because payments under $10.00 are free [21]. Dwolla buttons can be created in a similar way to the PayPal buttons only instead of using the recipient’s email we need a token. This token has to be generated by the content creator. Instructions for doing so are available on Tipsy’s landing page (see section 3.7). In effect, the content creator has to generate the button on its own from within Dwolla. The token is not a secret as any Dwolla button placed by a merchant has the token in its source. Tipsy will recreate the Dwolla payment button using Dwolla’s format, and all it needs is the token. After the user clicks on the button, they are redirected to Dwolla to make the payment. After the payment the user is redirected back to Tipsy, and the information about the success of the transaction is returned to Tipsy. Tipsy currently does not have a mechanism to handle failure of the redirect. Additional work has to be made so that in case of redirect failure, there is some other way to mark the visit as paid.

### 5.4 Storage

In this section I will describe where the log and settings are stored.

To respect user privacy Tipsy has no server (besides for the anonymous data gathering (see section 7) so the storage has to be in the browser. Chrome offers a number of different storage options. The most obvious for Tipsy’s purposes would be to use the simple local Storage
which is an HTML5 storage [17] within the browser. It has the advantage that it is very easily accessible to the user and browser independent. However, it only stores strings (so serialization needs to happen on the extension’s side) and there is no (clean) way to extend the maximum 5 MB of storage it allows for each domain. Another drawback is that it doesn’t automatically synchronize between browsers for a single user. There are workarounds but it does not have the native ability to do so.

The option that was settled upon was chrome.storage.local [18] for the following reasons. First of all, JSON objects can be stored which makes it a bit easier to use. It also stores a maximum of 5 MB but this maximum can easily be extended if needed [18]. A final benefit of chrome.storage.local is that it allows for the code to listen to any change events to it using chrome.storage.onChaged. The drawbacks of using chrome.storage.local are that it is not as easily accessible to the developer, so some developer ease is lost, and that interactions with it are asynchronous, so the code just has to deal with this. However, since the advantages outweighed these issues, chrome.storage.local was chosen. In any event, Tipsy’s code allows for an easy change of storage engine.

After running Tipsy for a month it was observed that the storage had used about 100kb of storage out of a maximum of 5MB - about 2%. Assuming that user browsing patterns are constant, the user will never delete their browsing, and that the growth is linear (which is reasonable since each visit adds the same amount of data) this suggests that after 50 months (just over 4 years) Tipsy will exceed its alloted memory. There are a number of ways of dealing with this. One way would be to simply delete old information. This is possible but would detract from the user experience since the old log information would not be visible to them anymore. It would however alleviate privacy concerns. Another option is to compress the log. This would be a good idea and would hopefully last long enough until Google increases the maximum storage allowance. A third option is to simply override the limit and make it limitless, which Chrome allows [18] by setting the storageArea permission...
to unlimited storage. If there are no detrimental effects of using a lot of the storage this would be the best option if privacy is of no concern. However, it is not a good idea to grow the storage indefinitely so a combination of these approaches would most likely be the most optimal solution.

5.5 Author Information

In order for a content creator to accept payments it will have to provide some identifier and payment method preferences. This information will have to be placed somewhere so that Tipsy can read it, and a format is needed so that Tipsy can reliably and consistently parse it. In this section I will describe this format and the way Tipsy parses it.

There are two methods to allow content creators so tell Tipsy how they wish to be paid. Content creators can either place a file, called tipsy.txt, at the root of their domain which would contain the information Tipsy needs to let users pay them. They can also place a link tag in a page’s source with this same information.

There are a number of advantages to using the tipsy.txt method. Adding a file to the root of the domain does not require any changes to the source of the pages referred to in the tipsy.txt file. Changing the payment information is also made easy since all the content creator needs to do is modify the corresponding entry in the file. All of the Tipsy data is stored in one place (and could be useful for eventually adding more data, see section 10.4).

However, some content creators might not have access to the root of the domain their content is hosted on but still would want to be able to be paid by Tipsy users. Some might want to be able provide this information on a more flexible, page by page, basis. For this reason, it is also possible to use a link tag. This method, however, requires a change to the source of the page, which some content creators might not want to do. Additionally, for sites with many subdomains where the content creators want to place a single payment information for all
the pages, it might be cumbersome to place a link tag on each page. For these situations using tipsy.txt might be a better option. Tipsy allows both.

5.5.1 tipsy.txt

The tipsy.txt file, similar in spirit to the robots.txt, contains the information the content creator wishes Tipsy to know. Although it is placed at the root of the domain, it still allows the content creator to specify different payment information for different subdomains. The standard used is yaml since it supports nested data, which is useful for specifying different information for different subdomains. The format looks as follows (fields in brackets are to be replaced with values):

version:
  [version number]

name:
  [name]

cache-duration:
  [cache duration amount] [cache duration unit]

payment-methods:
  [url-prefix 1]:
    [processor 1]: [info processor 1]
    [processor 2]: [info processor 2]
  [url-prefix 2]:
    [processor 1]: [info processor 1]
    [processor 2]: [info processor 2]

The version number tells Tipsy what version of the tipsy.txt format is in use. This is useful for allowing Tipsy to update the format while still supporting older versions.\(^4\)

\(^4\)Tipsy is currently on the 2nd version, 0.0.2
Under payment-methods the content creator can provide the payment methods for as many subdomains as they wish. The url-prefix could be something of the form “mycontent.html”. The processor could be “paypal” and the info processor could be the PayPal email “name@paypal.com”. To specify the entire domain the user can place a “_” for the url-prefix. The name, which is optional, allows the content creator to provide a name. Since looking for the tipsy.txt file requires making an HTTP request, making a request on each site visit might slightly affect the user's browsing experience. Therefore, if Tipsy finds the tipsy.txt file, it will cache it (in localStorage) so that the next time the user visits the site it will have the information in cache. In order to allow periodic refreshing of the cached tipsy.txt version, the content creator can specify a cache duration after which Tipsy will make another request to refresh its cached version. For sites that don’t have a tipsy.txt Tipsy will just cache that information (with a corresponding cache duration) as well so that it doesn’t make unnecessary requests.

Figure 17 shows an example of a tipsy.txt file. Tipsy's landing page (see section 3.7) provides content creators with information about the tipsy.txt format and a form that they can fill to automatically generate the file.

```
version: 0.0.1

<table>
<thead>
<tr>
<th>cache-duration: 3d</th>
</tr>
</thead>
<tbody>
<tr>
<td>payment-methods:</td>
</tr>
<tr>
<td>mysite.html:</td>
</tr>
<tr>
<td>paypal: <a href="mailto:me@mail.com">me@mail.com</a></td>
</tr>
<tr>
<td>dwolla: kRrCDD32hkwHE7ldn/xeI/WCQKqR</td>
</tr>
<tr>
<td>_:</td>
</tr>
<tr>
<td>stall: <a href="mailto:someone@else.com">someone@else.com</a></td>
</tr>
<tr>
<td>dwolla: H6E7ldH6E7ldKqR/KqRD32hkw</td>
</tr>
</tbody>
</table>
```

5.5.2 link Tag

The link tag idea was originally proposed by Dalton Hubble. This method is used for content creators today to lay authorship claims on their work. In fact, Google used to parse these tags to present authorship infor-

---

5 The request happens after the page’s content has loaded so the effect might be negligible.
mation in their search results only to have stopped doing so due to low adoption [19]. They are simple but still provide the necessary functionality that Tipsy needs. With link tags the content creator can just include the payment information in the head of their source as follows:

```html
<link rel="author" name="Me" data-paypal="me@paypal.com"/>
```

Notice that we use the `data-*` construct which is allowed by HTML for custom data attributes. While this is technically to be used for private data related to the page or application [20] it suits our purposes well.

This information can now easily be read by Tipsy by making a simple attribute query. The link tags can be gotten by something like:

```javascript
$('[rel="author"]')
```

After looking for the link tag that has the `rel='author'` we can, for example, get the PayPal payment information simply with something of the form:

```javascript
DOM.getAttribute("data-paypal")
```

A similar call is made for the other payment processor identification information. Using this format allows Tipsy to quickly and easily pick out the payment information without placing too much burden on the content creator. Tipsy will store this information in the log (see section 5.1) along with the other browsing data.

Content providers might want to use both the tipsy.txt and the link tag methods. Some domains might have subdomains where they have outside, frequently changing, authors write content and for whom they don’t want to have to modify the tipsy.txt each time the author and payment information change. Tipsy could then first look to see if the information is in the tipsy.txt file and only upon failure check the source.
From a security standpoint, both the tipsy.txt method and the link tag method have
issues. The tipsy.txt is vulnerable to a man-in-the-middle attack where an attacker’s
tipsy.txt might be served instead of the real one. The link tag is vulnerable to extensions
that could modify the page’s source and change the information in the tag. For a more
complete description of Tipsy’s security concerns see section 9.

5.5.3 RDFa

Initially I thought to use RDFa as the format to allow content creators to specify payment
information. RDFa’s built-in functionality for gathering machine readable metadata from
(X)HTML files seemed to make it the obvious choice. However, I think that including RDFa
might raise the barrier for content creators since including them requires some extra skill, so
I didn’t choose it for the first implementation. RDFas are a good choice when one wants to
include complex, hierarchical metadata that could be easily read by web crawlers or other
machines. Still, for cases where the author information is not immediately obvious, or where
a lot of different authors are represented in a single source RDFa might still be the preferred
method. Eventually RDFa parsing ability can be added to Tipsy for the content creators
that wish to use that format.
6 System Architecture

In this section I will describe Tipsy’s overall structure. Tipsy is a Chrome extension and some of its design reflects the architecture favored by Chrome extension development. See Figure 18 for a full picture of Tipsy.

In the extension there are are two main scripts that run: (1) the Background Script [22] and (2) the Content Script [23].

6.1 Background Script

The Background script runs from the moment Tipsy is installed. It holds the bulk of the extension. The Background Script is responsible for logging, for generating the User Interface when necessary, for interacting with storage and for interacting with the payment processors. Most of Tipsy is run from the Background Script. The Background Script has the ability to use any of the Chrome APIs discussed so far.

6.2 Content Script

However, the Background Script is unable to interact with the pages the user visits. This makes sense from a safety and security standpoint. Its scope is limited to the Tipsy extension domain. But Tipsy needs to be be able to interact with the page as it needs to read the author information which is stored in the page’s source. To do that we use the Content Script. The Content Script runs in the context of the page that the user is visiting. It cannot use almost any of the chrome.* APIs (one exception is the chrome.runtime.* API) nor can it directly manipulate anything in Tipsy’s domain’s scope. The main responsibility of the Content Script in Tipsy is to parse author information that might be in the page in the form of a link tag.
As mentioned in section 3.7.2, Tipsy allows content creators to check whether a visitor has Tipsy installed. This is implemented by having the Content Script place a specific Tipsy DOM element in the page’s source. The library provided by Tipsy to check for Tipsy looks for this element.

6.3 Message Passing

The author information has to be stored in the log so it has to be made available to the Background Script. The Content Script uses Message Passing to achieve this. Message Passing is a way for threads from different domains to talk to each other and it lends itself naturally to our use case. The Content Script posts messages to the Background Script whose content contains the author information. These messages can simply be sent with chrome.runtime.sendMessage.

6.4 Rich History

In effect, the log that stores the user’s browsing history and the code that implements it provide an augmented browsing history functionality, or Rich History. All of Tipsy’s state (aside from settings) is stored as the Rich History and is used by Tipsy to calculate payments. This Rich History functionality could be thought of, and packaged, as a unit separate from Tipsy’s core responsibility of calculating payments. It could then provide an API for use by other applications, such as Eyebrowse [24], that need the same functionality.
Figure 18: An overview of Tipsy’s system.
7 Data Gathering

In order to learn from users and improve Tipsy, Tipsy is able to send some browsing information from the users to a server. The information sent is completely anonymous and Tipsy is unable to distinguish users from each other. One goal of this data gathering is to permit Tipsy to inform particular content creators of how much money they could have been making had they been using Tipsy and thereby encourage Tipsy’s adoption. The data sent includes the time spent per site and the estimated amount of money a site would have made. On the server this data gets added to a global count per site.

The data gathering feature is disabled by default and the user must opt in to enable it. The user has the option to do so during setup or later.
8 User Studies

I ran a study on a number of users and incorporated some of the results in Tipsy’s design. In this section I will describe the results of the user studies. There were two main parts to the study: (1) whether users liked the idea at all and would actually install Tipsy, and (2) what their reaction to the interface was. I first explained to them what Tipsy was and then had them go through the initial set-up. I then had them visit tbranyen.com because that page has real payment info. Then I changed their rates to large numbers so that they would generate an amount on the Contributions Page. I noted down all the comments that they were making while they went through the process and the points where they seemed confused.

8.1 Tipsy Idea

Around half of the users said outright that they would not install Tipsy at all because “why should [they] pay for something that [they] could get for free”. One of them first assumed that Tipsy was implementing some kind of paywall because “why else would anyone pay”. This reaction was not surprising. The Radiohead release also had about 40% of people who downloaded for free (see section 1.2), and an album, unlike most internet content, is not something that is traditionally downloaded for free. What was interesting was the reaction of the other 50%, which was split into two groups. Some wanted to “sign up” immediately because there were sites they wanted to pay for. Others would use it only if it got rid of ads. We learn from this that some users who are willing to donate still need an incentive to pay, which makes it more crucial that Tipsy give the user something more (like getting rid of advertisements) than satisfaction for supporting online content. Using the more imperative language of “Pay” instead of “Donate” and making Tipsy a standard would also likely help in getting the more hesitant users to use Tipsy.
8.2 Tipsy Interface

I will describe the users' reaction to each of the different pages in Tipsy. You can see the screenshots of the different pages in section 4.

Getting Started

Most users were able to get through the set-up pretty easily. One of the users was confused by the reminders. They didn’t understand why they needed to choose a reminder. They assumed Tipsy would just decide automatically when to remind them.

Log Page

Some users were confused by the “Days with Visit” column, and one suggested to change it to “Number of Days with Visit”. It is not clear whether that column should be kept at all so perhaps it should be hidden until a user asks for it by selecting an option. None of the users realized that it was possible to click on a row to expand the URLs to see all the visits. This is why I made sure to change the feedback so that the mouse icon is changed to a “pointer” which suggest that you can click under it. Once I showed them how to expand a row one of the users commented that there were too many entries. I do agree that sometimes the number of rows gets too large but it is not obvious in what way the individual visits should be grouped. Grouping them such that all visits within an hour are combined into one row seems like a plausible solution. Another solution would be to combine the visits by page and then rank them by amount of time spent. In any event, the Log Page is not the main point of Tipsy, but is meant to give the users the information if they wish to find it. One user made an interesting comment that the “clear history” feature shouldn’t delete all the history. They felt that there should be a more fine-grained option of deleting specific entries. I agree with this and it is currently being implemented.
Settings

The users’ questions regarding the different rate types suggest that either a better way of explaining it is necessary or that Tipsy should provide just one option. All the users needed more explanation about the difference between the Calendar and Activity Rates. One user thought that the Activity rate would track all tabs that are open at the same time even if the user is not focused on it. It is possible that since the Calendar Rate follows the “Subscription Model” it should be the only one displayed. An “Advanced” option would be there for the Activity Rate. However, after it was explained there was not consensus about which rate was preferred. Two of the users felt that the estimate provided by the Activity Rate was useful. One user suggested that the wording “Set a Contribution Goal” should be changed to “How Much do you Want to Pay”.

Contributions Page

This page was immediately understood by all of the users. Users particularly liked the fact that they were redirected to the payment processors page to complete the payment since it made them “feel more secure about the payment”. It seems that the familiar ground of PayPal puts users at ease about making payments so this behavior should be kept. One user noted that the page should have a big “Pay Now” indication telling users to pay because they thought the current Contributions Page does not contain forceful enough language to compel payment.

8.3 General User Comments

There were some general remarks that some users made.

One user noted that the idea of periodic payments should be more strictly followed and that there should be a “payment time” where you decide for each entry in the Contributions table whether you want to pay or not. At the end of this “payment time” all amounts would be
clear. This user felt that otherwise it just feels like a “never ending cycle of payments”.

One user thought that the order of the tabs should be, from left to right, Contributions, Log, Settings instead of Log, Settings, Contributions. I see what this user means since it is common in many applications to have the main part of the program first and the settings at the end. In Tipsy, however, the current order suggest the flow from browsing to calculation to payment.
9 Security Risks

Since Tipsy deals will actual user payments it is especially necessary to protect against potential security issues. In this section I will discuss some of Tipsy’s security risks and how they can be mitigated.

Source Modification

The main risks in Tipsy involve making payments to the wrong entity - which will occur by making some modification to or preempting the payment information that the content creators provide to Tipsy. If a content creator is using link tags, since theses tags are in the page’s source, any extension (or other malicious code) that can manipulate the source before Tipsy is able to parse it can replace the original payment identifier with their own. In today’s browsers it is not easy for arbitrary malicious code (eg from an iframe) to modify the page. The exception (of which Tipsy itself takes advantage of) is a browser extension. They are able to modify the pages at will. When the user installs an extension the browser will warn the user about what permissions the extension is taking, but it is unclear how much this deters users from installing anything. Of course, in that case the problem is reduced to a more general problem since any extension can redirect payments by making appropriate modifications to the page (eg. a banking page).

This problem could be fixed by authenticating content creators. The content creator could place a digital signature signed by Tipsy in the link tag which Tipsy could then verify. This however, just repackages the problem since it is not obvious how the key exchange would happen in the first place.

Man In The Middle

Using the tipsy.txt file could prevent the above problem. As mentioned in section 5.5.2, content creators could use a combination of both a tipsy.txt file and a link tag. Tipsy
could also allow the content creators to specify in the tipsy.txt file that Tipsy should not take any link tags into account. Instead, all the payment information should come only from the tipsy.txt file in the root of their domain to which (hopefully) malicious agents have no access. Tipsy will get the information by making a GET request to the content provider’s server. There is still however, the possibility of another attack vector: the Man-In-The-Middle attack. A malicious DNS server (or mobile router or hotspot\textsuperscript{6}) can respond to the query with its own tipsy.txt. It could even generate a copied, authentic-looking version of the site the user is requesting so that the user might not even realize that they are on the wrong site. If the content provider uses https then this attack vector is harder since the browser will do the authentication for Tipsy. However, not all content providers have certificates so this problem remains. Since Tipsy caches the tipsy.txt data (see section 17) the malicious payment information would remain even after the user goes to the real content at a later point (the attacker can even specify a particularly long cache-duration).

**Author Verification**

Another solution to the authentication problem could use something similar to what Google does to verify domains \cite{google} where they ask the user to place an HTML file, with contents that Google specifies, within a particular time interval. If the user is able to do that it is assumed that they are the authentic owner of the given site. Tipsy could use this method, along with a digital signature, with the tipsy.txt file. Content providers would sign up with Tipsy at tipsy.csail.mit.edu and Tipsy would provide with a digital signature to place in their tipsy.txt file which they are to upload within some time interval. If they do, and the signature is verified, then they are authenticated and the information in the file is assumed to be true. From then on whenever a new tipsy.txt file is fetched, Tipsy will verify the signature. This could allow Tipsy to detect of the tipsy.txt file that is served is authentic and solves the initial key-exchange problem.

\textsuperscript{6}A badly behaving airport WIFI hotspot actually brought this problem to light.
Accountability

Since payments are currently made through either PayPal or Dwolla there will always remain a record of how much was paid to whom, so whenever it is realized that payments were being made to a malicious account the recipient can be found and potentially held accountable.

When Tipsy incorporates other payment methods that provide more anonymity (eg. bitcoin, see section 10.3), it will be much harder to track down to what account the money was paid.
10 Future Work

There are a number of future improvements to be made in Tipsy. In this section I will describe some of them.

10.1 Firefox

Tipsy was chosen to be developed for Chrome for two main reasons. First, because Chrome is the most popular browser. Studies of browser adoption statistics [26] show that almost 64% of the browsers used are Chrome. Second, Chrome provides an easy-to-use and (relatively) well documented API for extension development. However, after Chrome the second most popular browser is Firefox (at nearly 22%). It would clearly help spread Tipsy usage if it were available for Firefox (and thus 86% of online users). While not as streamlined as Chrome's, Firefox has a well supported extension\textsuperscript{7} API.

Work on a Tipsy Firefox extension has been started but it currently lacks the full functionality necessary. Even though much of the code is shared between the two - Tipsy is mostly HTML/CSS/JavaScript - some of Tipsy's core functionality uses the Chrome APIs (eg. for the tracking, see section 5.1).

10.2 Mobile

While users do a lot of browsing from a desktop browser an increasing amount of time spent consuming content is done from mobile devices. Unfortunately, Tipsy is currently unable to tap into that browsing because Chrome extensions currently do not run on mobile platforms. There are several ways to allow Tipsy to work on mobile:

\textsuperscript{7}Technically Firefox extensions are called "Add-ons".
Mobile Browsers with Extensions

The most obvious solution would be to have mobile browsers that support extensions. Some mobile browsers already do. The Dolphin browser [27], for example, supports extensions so a Tipsy extension could be developed. A nice feature of the Dolphin browser is that it can synchronize data across multiple browsers so it could work in conjunction with the Tipsy’s Chrome browser extension. It is likely that in the future more mobile browsers will support extensions so this solution would be the most natural.

Tipsified Mobile Browser

Instead of waiting for browsers to support extensions Tipsy could also provide a mobile browser that has Tipsy built in natively. It could be developed to sync with the user’s regular Chrome browser. While this method could work, generating significant adoption might be challenging since it would require mobile users to stop using their regular browser and switch to the Tipsified one.

App

A Tipsy app could be created that would have a similar functionality to Tipsy’s browser extensions. All the user would have to do is install the Tipsy app, keep browsing with their regular browser and let app would do the rest. While user friendly, this solution might be difficult to implement. It is unclear how easily this app will be able access to the information necessary to track browsing activity. This method would not require users to stop using their regular browser. Of course, to reach a maximum amount of users, at least both an Android and an iOS app would have to be developed.

Script Loading

Another option to capture mobile browsing would be to have a Tipsy script that could simply be run from any content creator’s page. The content creator could include this script in their
source and Tipsy's functionality would be available.

An advantage of this approach is that it decouples the way the user is browsing from the actual content being consumed since Tipsy will be included in the content itself. This way, a separate extension or app need not be developed for each browsing method.

A way to implement this would be to split Tipsy into a separate tracking part and payment part. The tracking part would be embedded in the content creator’s source and would open an iframe to tipsy.csail.mit.edu where the user’s data could be stored within the browser. The payment part would be within the tipsy.csail.mit.edu iframe and the user can navigate there to pay.

10.3 Additional Payment Options

Additional payment processors (like Stripe) or frameworks (like bitcoin) could improve Tipsy’s adoption. So far we have included PayPal because of its ubiquity and dwolla because its payments under $10.00 are free of transaction costs.

Bitcoin would especially be nice because it would allow content creators an additional layer of anonymity above the regular payment options. Dwolla and PayPal can be made anonymous to users if a particular account does not provide information about its owner. But both PayPal or Dwolla (and thus potentially others) would know who the owner is. With bitcoin a much stronger anonymity could be guaranteed.

Once more payment options are available Tipsy’s Contributions page (Figure. 14) will ideally not show too many options. Hiding the payment options the user does not use would be a good choice. In the current implementation Tipsy shows all the options the content creator accepts since the maximum it will show is two. Showing the ones that the user does not currently have set up is useful as an encouragement to the user to set them up. It would perhaps be preferable to have the buttons for the payment methods the user does not use to
show up smaller such that they both don’t take up too much visual space but still remind
the user which payment providers they might want to install. Another option would be to
provide a button the user can use to show or hide the sites that accept payments only in
methods the user has not set up.

Additionally, in the current implementation content creators who want to accept payments
with dwolla need to provide their key in the link tag or tipsy.txt file. While Tipsy’s
landing page does provide information on how to generate one, it would be easier for the
content providers if the Tipsy landing page itself could generate it for them or, preferably,
if they don’t need the key at all but can just include a dwolla email or username.

10.4 Feedback Model

Currently, content creators can check to see if a visitor to their page has Tipsy installed (see
section 3.7.2). However, a more complete feedback model would potentially be more welcome
to the content creators. A scenario could be imagined where a content creator would like to
display a message to the user thanking them for supporting their content. Perhaps they can
even tailor their messages based on how much money the user has paid them. Tipsy could
perhaps also encourage a non-paying Tipsy user to start paying by reminding the user how
much time the user spent on the site.

All of this would seemingly require that the content creator somehow have access to to the
user’s browsing activity. This might not be a good idea from a user privacy standpoint since
we wouldn’t want the content creator to have access to all of the user’s Tipsy Log. The user
might not even want a given content creator to have access to the information in their log
from that content creator. Another way to get this functionality would be to have Tipsy
itself place the information in the page for the content provider. The content provider can
place an empty DOM element in the page with a special Tipsy attribute so that Tipsy can
fill it out with the information that the content creators tells Tipsy to place there. The
content creator can tell Tipsy what to place through an additional field to the tipsy.txt file. The information does not necessarily have to be free-form. Tipsy can specify different feedback message types from which the content creator can select and specify through the tipsy.txt file. This way the user’s data is never sent to the content creator, although whatever information is displayed on the page would be available to the content creator’s site.

A way to allow content creators to provide this feedback while making it invisible even to the content creators themselves could be by having the Tipsy extension provide an iframe to which the sites can send messages. The content creator could then send a template to the Tipsy iframe through message passing. Or it could choose from a number Tipsy provided templates by making a request to the iframe and specifying the template through the URL. This way Tipsy will know what feedback from the content creator to present to the user without having the content creator know the actual data.

10.5 Ad Removal

Once Tipsy’s adoption is significant, it could naturally replace the ad model that sites use to fund themselves. Instead of showing ads, a site could consist of content only since it will be funded by Tipsy users. The content creators can check if the user has Tipsy using the provided library (see section 3.7.2).

There are two ways content creators could handle Tipsy users. One would be to simply not serve ads to any user that has Tipsy installed. This would effectively turn Tipsy into an ad-blocker since just installing Tipsy would stop those sites from serving ads. While not optimal, this system could work since there already are ad-blocking extensions a user could download [2] and Tipsy would just be one that also happens to allow users to pay. Another option would be to not serve ads to Tipsy users who already have made a contribution. While this might seemingly prevent users from using Tipsy as an ad-blocking tool, users can
lie to Tipsy about making payments so there is no way for Tipsy to know for certain whether
the user has paid. This would then reduce to the first option.

10.6 Multiple Authors for Single Content

Currently Tipsy assumes that a there is only a single author per page. Both the tipsy.txt
and the link tags only allow authors to specify a single payment account per page. There
is no reason this has to be the case. It would be interesting to allow authors to specify
multiple payees for a page and have the payment divide between the different payees. Tipsy
could allow the authors to also specify exactly how the the payment should be split - should
both get half the amount or should a different division occur. Of course, this would have
to change how the payment buttons are generated and how the user is guided to make
payments. Neither PayPal nor dwolla allow for buttons that pay more than one payee. This
could be solved by implementing what is discussed in the next section.

10.7 Avoiding Too Many Transactions

As mentioned in section 3.5, the current implementation of Tipsy requires a separate payment
session for each payee. Instead of just hitting “pay” a user has to go through as many payment
sessions as there are payees, which could be detrimental to the user experience. Additionally,
if payments are being made through a payment processor that charges transaction costs, each
separate transaction will come with its own transaction cost which could lead to some cases
where a non-trivial amount of money is owed but cannot be paid. In this section I will
discuss some mechanisms to solve this problem.

Lottery

In the case described in section 3.5 there are 20 payments all of $0.30 to be made with
PayPal. Since PayPal takes a $0.30 cut, this means that there are $6.00 owed by the user that cannot be paid. A way to solve this would be to assign each of the payees a virtual “lottery ticket” and then randomly select one of them. The payee whose lottery ticket is chosen will get $5.70. While this user in the end only pays one of the payees, averaged over all users and all content creators, the lottery provides (almost\footnote{The single $0.30 is still missing.}) the same payment result that would have been reached in the ideal case with no transaction costs.

Clearing House

Another, perhaps more general, way of solving the problems that come from having to make multiple payments involves setting up a clearing house. A clearing house would be a trusted organization to whom the users pays their entire bill in one payment. The clearing house is then responsible for distributing the individual payments to the content creators. Content creators would be members of the clearing house. Tipsy would have to be changed to not only initiate payments but to also send information about how the payments should be divided between the various recipients. While the clearing house would have to avoid running into the same issues with transaction costs when it makes its individual payments to the content creators, the clearing house could often have instructions to pay the same payee from multiple users so it can combine all of those payments into a single transaction. The clearing house would also play a part in making Tipsy a standard method for paying “the internet”.

10.8 tipsy.txt location

An RFC [28] has gone out from the IETF that proposes a standard for where to place “Well Known URIs” like robots.txt. They suggest placing these types of files in paths with a special prefix, "/.well-known/". The Tipsy landing page could suggest that tipsy.txt be
placed there. Tipsy will check "/.well-known/" before checking the root.
11 Conclusion

Microdonations, when presented in a certain manner, could be a way to improve the internet’s business model. By making it easy for users and fair to content providers, an environment could be created such that ads are unnecessary, thereby freeing the internet from unwanted distractions and simplifying it to content alone. Tipsy is a novel approach to microdonations with this goal in mind and will hopefully make the internet a better place.
References


63


[18] https://developer.chrome.com/extensions/storage#property-local


[21] https://www.dwolla.com/fees

[22] https://developer.chrome.com/extensions/background_pages

[23] https://developer.chrome.com/extensions/content_scripts


[25] https://support.google.com/webmasters/answer/35658


[28] https://www.ietf.org/rfc/rfc5785.txt