Delft University of Technology

Contextproject - Blockchain

An Embedded, Interactive Explorer for the Tribler Network

■-■-■- BlockchainBoys -■-■-■–

Yinghao Dai, Max van Deursen, Geert Habben Jansen,
Bram van den Heuvel, Ruben Keulemans, Tim Speelman

supervised by
Martijn Gribnau, Stefan Hugtenburg, Johan Pouwelse and Alberto Bacchelli

June 29, 2017

Abstract

Tribler is an open-source BitTorrent client for anonymous peer-to-peer file sharing. It attempts to solve the Tragedy of the Commons problem by recording up- and download history of peers. Based on this history, it may refuse service to users who don't contribute. If users do not understand these mechanisms, they will not change their behavior. To this end, we have added a feature to Tribler: The Network Explorer. It attempts to explain the mechanisms and incentives within the Tribler network by visualizing the network and collected statistics.

Contents

1 Introduction **1**
- 1.1 The Tribler Project . . . 1
- 1.2 Goal of this project . . . 1
- 1.3 End-user's requirements . . . 2

2 Product description **3**
- 2.1 An Overview of the network explorer . . . 3
- 2.2 The features not implemented . . . 5

3 Code quality description **6**
- 3.1 Legacy code base . . . 6
- 3.2 Our code quality . . . 6
- 3.3 Shipping our code within Tribler . . . 7

4 Components of the Network Explorer **8**
- 4.1 Front-end code structure . . . 8
- 4.2 Back-end code structure . . . 8

5 Evaluating Usability and Awareness **10**
- 5.1 Evaluation method . . . 10
- 5.2 Results . . . 11
- 5.3 Conclusion . . . 11
- 5.4 Discussion . . . 11

6 Evaluation of functional modules **13**
- 6.1 Continuous Integration . . . 13

7 Further development of the product **14**
- 7.1 Actual Trust . . . 14
- 7.2 Relevant nodes . . . 14
- 7.3 Alternate views . . . 15

A Evaluating Usability and Awareness **16**
- A.1 Debriefing questions . . . 16
- A.2 User feedback . . . 16
  - A.2.1 Feedback gathered in the first round . . . 16

A.2.2 Feedback gathered in the second round 17

Chapter 1

Introduction

1.1 The Tribler Project

Tribler is an open-source BitTorrent client for anonymous peer-to-peer file sharing (Pouwelse, 2008). It is fully decentralized, protecting its users against censorship, lawyer based attacks and government control. Like many decentralized systems, it suffers from a problem known as the Tragedy of the Commons. As described by Hardin (1968), individual users of a shared unregulated resource may act only in their self-interest, posing a threat to the existence of the resource. In the context of file sharing, the resource is kept alive by users uploading their files. Users who do not contribute are called free-riders. In order to protect the resource, systems must find a way to identify them and deny service.

Since Tribler users are anonymous and creating a new identity is virtually free, identifying free-riders is more difficult. Tribler uses TrustChain (Norberhuis, 2015) to capture interactions between users. This establishes the notion of reputation of a peer in the network. Although users cannot directly tamper with these interactions, it is susceptible to Sybil attacks. In this kind of attack a user creates a set of fake identities, and fakes transactions between them, to improve his own reputation. In an attempt to detect these attacks, Otte (2016) has investigated the usage of the Net-Flow and Temporal PageRank algorithms to detect Sybil attacks. Based on the aforementioned technologies, a trust level of peers can be determined. A peer is considered trustworthy if it has contributed sufficiently and is unlikely to be a sybil. Currently, these algorithms are not integrated into Tribler. When they are, Tribler can refuse service to other peers.

A first iteration of the service-refusing algorithm will use a simple metric in order to decide whether a user should be prevented from downloading content: if the balance (difference) between uploaded and downloaded data gets too negative, i.e. exceeds a given threshold, then the user will be considered a free-rider. This means that he or she has to start uploading before being able to download again.

Later iterations will expand on this binary model and use models of preference and limiting.

1.2 Goal of this project

As most of the Tribler users are unaware of these technologies, they may not understand why their service is being (partially) denied. This may leave them frustrated, and abandon the software

instead of improving their contribution to the network.

This gap between the Tribler software and the person using it must be bridged. To this end we developed a Network Explorer that shows users exactly what is going on around them, what their reputation is, how that came to be, how they can improve it, but most importantly, why they should improve it.

1.3 End-user's requirements

Since the product is an extension of Tribler, its users are existing Tribler users. Our goal is to offer existing users a new feature that enhances their Tribler experience. Current users need to understand how they obtained their current balance and how this affects their download speed. Moreover, they need to understand how to improve this balance. Users only contribute to the network once it is in their short-term interest (Hardin, 1968). Some of these features might also convince currrent users of other torrent clients of the added value of Tribler.

This understanding is furthered by allowing these users to gain insight into the network of their peers. This is done by showing, for example, the calculated balance of a node in the network, and relations between nodes. The users are then able to compare these different balances.

For this to work, the users need to understand what they are seeing in the Network Explorer. Therefore, there needs to be a help page with a clear description of what the different components represent. This has to be shown in such a way, that an average user without a background in computer science engineering can easily understand. While doing this, we have to ensure that the user experience does not decay. For instance, the start-up time of the Tribler application must not be negatively affected.

Moreover, seeing so much information about the network could make users worried about their own anonymity. We have to make sure that users do not doubt about this matter, and become fearful of using Tribler.

Chapter 2

Product description

When an end user downloads and/or uploads files using the Tribler network, he or she exchanges data with other Tribler users. In the background, Tribler keeps track of how much data was exchanged and between whom, not only of the end user but of all peers it can find. This way it forms an ever-growing picture of the Tribler network and the traffic between users.

2.1 An Overview of the network explorer

The network explorer is simply a view on the gathered data. It displays all users, their upload and download totals, and their exchanges with other users. It provides the end user with valuable insight about his or her position in the network.

As displayed in figure 2.1, users are represented as circles with a short anonymous pseudonym. The circles are connected with lines, which represent the amount of exchanged between users. An intuitive representation of the data is given through visual cues such as colors, circle size and line thickness. When the end user hovers over these circles and lines the exact quantities are displayed in a sub-window called the *inspector* (see figure 2.2).

Tribler judges its users based on these data and determines when to stop uploading to a specific user; i.e. when its 'reputation' is too bad. Determining a fair metric for reputation is difficult, but Tribler currently uses 'balance' for this. The balance is a user's total upload minus download. Reputation is presented through a color; a user with a poor reputation is colored red, a good user ("toffe peer") is colored green.

Displaying all known users in a single view would become impossible as the gathered data grows. To this end the network explorer only shows a portion it. It focuses on a single user (initially the end user), centered in the view, and displays its peers in a ring around it. A second ring contains all peers-of-peers. This is called a 'radial layout'. When a user has a lot of peers, only its biggest exchanges are displayed.

Any of the users (circles) can be clicked, which shifts the focus to that user.The clicked user's circle animates toward the center and its known peers and peers-of-peers are revealed on their respective rings. This way the end user can explore the network and inspect peers who are more than two steps away. By means of helping the end user find the way back to his or her own node, a 'Back To You' button is provided which triggers a step-by-step animation navigating all the way back to the user's own circle.

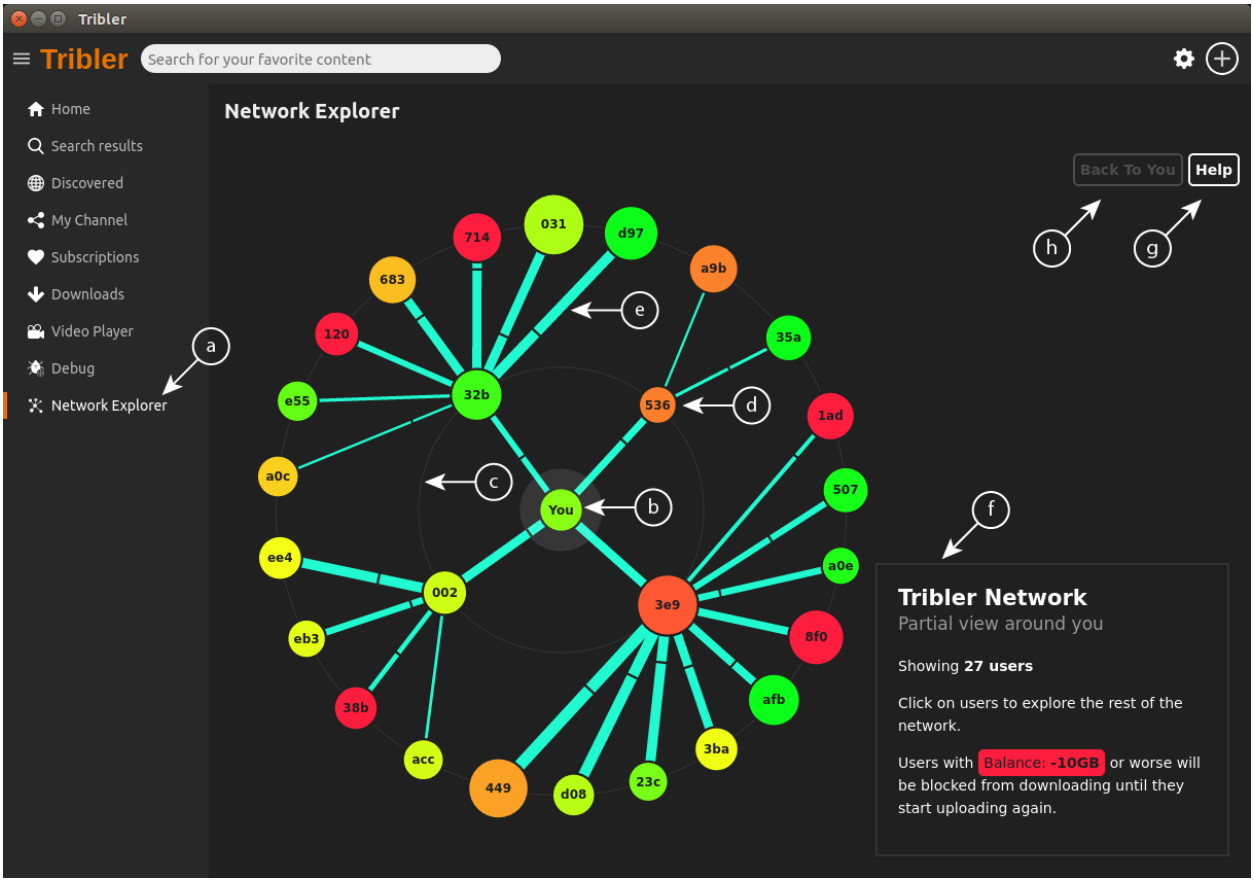

Figure 2.1: The network explorer feature inside Tribler with (a) menu-item, (b) end-user's node, (c) ring containing first level neighbors, (d) first level neighbor, (e) exchanged data between users, (f) inspector, (g) help button and (h) back-to-you button

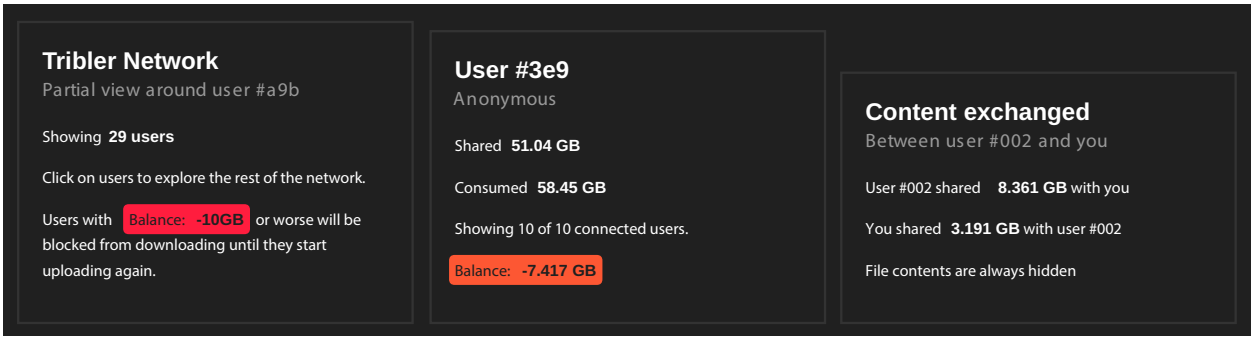

Figure 2.2: Different modes of the inspector, displaying information about the network (left), the hovered user (middle) or the hovered connection (right).

2.2 The features not implemented

The original intention of the network explorer, as described in the Product Vision document, was visualizing 'trust'. As it turns out, Tribler developers have not yet clearly defined their notion of trust. This makes it impossible to visualize it, for now. Nevertheless, there should always be a metric on which denial of service by Tribler is based. The network explorer shows this score by means of color. Even though calculation of the color may change, its message remains: a red user should worry about being blocked, a green user can rest assured.

More advanced calculations of trust call for more elaborate visualization. A notion of connectedness could be introduced, indicating how many users a particular user has connected to or how much data was exchanged. This could yield a view showing the different paths along which two users are connected, indicating size, number of middle-men of number of hops (shortest path between to users). Alternatively, it could yield a 'cluster' view where well connected users are positioned more closely together. This gives a sense of clusters; groups of users who have many connections with each other but few with users outside their cluster. A cluster view would be a great indicator of Sybil attacks.

Ultimately, a combination of views may be more effective than a single view. This can simply be solved by adding tabs or sub-navigation to the network explorer window allowing the user to switch between views.

Chapter 3

Code quality description

3.1 Legacy code base

The Tribler code base is not perfect[1]. Some issues are bad documentation or no documentation at all, very long methods and classes, inconsistent naming, old Python version usage and untested code. Because the Network Explorer is an addition to Tribler, and uses and extends its REST API, database and database communication classes, we are forced to work with this legacy code base. This has important implications including investing time in understanding the code we are using and extending, writing maintainable code without proper examples while being consistent with the conventions (if existing). Furthermore we constantly need to stay up to date with the Tribler fork of the Tribler repository. This takes a lot of time because new additions to the development branch can cause merge conflicts with our code, requiring several hours to create working code again in some cases. The above distinguishes the Network Display project from the other context projects, where software is developed from scratch, without the need to integrate it into an existing code base.

3.2 Our code quality

Working Agile using the SCRUM framework helped us write high quality code. Although we were integrated our product in an existing code base, we made sure to focus on the maintainability of our code. Each morning in the daily stand up meeting group members indicated issues and whether they needed need solving them. Pull requests were kept small and we used a "maximal one pull request at a time per person" policy. Per pull request group members reviewed the introduced functionality and code quality. Changes were adopted if needed. Moreover, in the last few sprints, the focus was on refactoring our code to be even more maintainable and neatly organized. This process lead to an overall outstanding code quality.

[1] Code base statistics on http://jenkins.tribler.org:9000/

3.3 Shipping our code within Tribler

Our project distinguishes itself from the other context projects since our deliverable is not going to be thrown away, but instead to be merged within the Tribler code base and used by millions of users. This lead to a difficult situation, since we had to comply with not only our assigned product owner and the context teaching assistant, but the Tribler developers as well. In order to comply to the latter party, we had to comply with multiple constraints and conventions in order to fit into the existing code base of Tribler. While doing this, we made sure that the code was easy to maintain for future modifications.

Chapter 4

Components of the Network Explorer

The Network Explorer consists of modules in both the back-end and the front-end. These modules communicate through a REST API.

4.1 Front-end code structure

The front-end of a single Tribler window is composed as follows. QT loads a Chromium based browser engine to display an HTML web page containing the actual Network Explorer view. The web-page loads all CSS and JavaScript modules, which compose the Network Explorer, and the required library D3.js, used for drawing and animation.

When the focus is moved to a user, the front-end sends a request to the back-end. It requests the clicked user as a 'focus node' and asks for its first and second level neighbors. When the response comes back, the DataProcessor does the necessary data conversion required by other front-end modules.

The processed data contains nodes (users) which must first be positioned. A positioning algorithm uses breadth-first traversal to make a tree from the graph. It then divides the space on the rings over all nodes. The animating of nodes to their desired positions is then done using D3 forces. A custom built 'torque force' ensures the nodes move along their own ring and not in a straight line.

A set of drawing modules takes care of drawing the users and connections as circles and lines. It binds to mouse click events for loading a different focus node and to mouse hover events for highlighting and displaying information in the inspector widget.

A separate module is responsible for guiding the user back through the nodes he or she clicked, using animated markers and careful timing. This ensures the user does not lose orientation.

4.2 Back-end code structure

The HTTP request from the front-end is received at an API endpoint in the Tribler Core. This request should specify the public key of the user it wishes to focus on. Additionally it may specify the number of neighbor levels and limits for each level. Finally, a list of 'mandatory' users may be provided whom may never be limited from the response.

An extension of the Community module first queries all users with connections to the focus node. It filters out the smallest connections (lowest up-/download) if requested. This is the set of first level neighbors. A second query lists all connections from users in this set with other users. This is limited in the same fashion and the process is repeated for every requested level. The data is converted into a suitable format and sent back to the front-end.

The database contains a table with all known transactions between Tribler users, not only with the end user but also between others. A crawler included with Tribler is responsible for scanning the network and filling this database.

The transactions table is in part encoded by a Python script which makes it unsuitable for querying sums. As a solution, our product comes with a custom aggregates table, listing pairs of users and the total exchange between them. Whenever a new transaction is added to the database, the aggregates table is updated as well.

Chapter 5

Evaluating Usability and Awareness

The Network Explorer is an addition to Tribler which helps users understand the Tribler Network by letting them interactively explore using the mouse. User actions should be self evident so all attention can be used to understand the visualization. From the start of the project we developed and designed with this goal in mind.

During the first six weeks of development, the group members and Product Owners Johan and Stefan gave their feedback and opinion about the usability. In the seventh week the first round of user testing was performed. Testing with users is important to verify if the assumptions made are correct. After processing the feedback the product was adjusted and the week thereafter a new user testing round followed.

5.1 Evaluation method

On the basis of the gathered information mentioned above, we set up a user experiment in order to test our product against usability and awareness. Using this test, we wanted to check whether users, with or without a computer science background, could easily understand, use, and know the importance of our product.

When choosing the target audience for participating in our test, we took into account the likelihood that a given person would use Tribler. After all, testing a Tribler component on a person that will never use Tribler, makes little to no sense. Therefore, we asked five people from another groups in our context, three people from other contexts in our course, and two mathematics students.

Our evaluation was performed in two iterations, which meant we executed the test, thereby generating results, evaluated and analyzed these afterwards, before improving our product and starting a new round of evaluation, executing another test, etc.

The test execution procedure was as follows:

1. We started explaining the procedure to the participant, thereby stating that we were recording the audio during the tests, and would annihilate all recordings within ten working days. In fact, only one of the audio recordings had been listened to afterwards, as we made useful notes during the tests as well.

2. Moreover, we explained that we wanted the participant to "think aloud" (and how to do

this) during the usage of our product, and that we would ask some additional questions afterwards.

3. At the end, if the participant wished to, we could explain more about Tribler and our product, so that the participants would not go home with the unease of having unanswered questions.

4. After this, the execution of the test took place, exactly as described above.

That means we conducted a minimal-risk test, which we verified using the ethics checklist of the TU Delft Human Research Ethics Committee. We did annihilate the recordings made in the first round, and even refrained from making recordings in the second round, as we felt that the notes sufficed.

This decision followed from the analysis that we made after the first round. When setting up the test procedure, we decided that one person would accompany and question the participant, and a second person would make notes. However, there was a possibility that one of us would miss information or interpret the saying differently, which meant having a recording and being able to listen another time, would be a blessing. However, as this happened only once, this did not outweigh the risks of recording for the participants anymore. Therefore, we decided not to record the audio in the second round.

5.2 Results

The results of the user tests can be found in the appendix. The feedback as well as the product adjustments following from this are stated. Usability and awareness feedback are not distinct because of their large overlap.

5.3 Conclusion

Based on the test results, we can conclude that users can indeed quite easily understand and use our product. This was the main encountered problem during the first round (the comprehensibility for users that are not familiar with Tribler or torrenting), and had been greatly improved before the second round. The awareness of our product's importance, however, turned out to be slightly insufficient in the second round as well. After the improvements of the second round, we believe this issue has been addressed. Unfortunately, there is no third round of user testing in order to verify this.

5.4 Discussion

Having such a third round, would naturally have been an improvement. Besides, if we started the user tests earlier, we would have done less redundant work. Moreover, we could have asked other groups of participants than only university students, since they are potential Tribler users as well. Having more participants anyway, would have helped as well, in creating a better view of how the users appreciate our product.

The main limitation of our evaluation procedure, was the procedure itself. By explaining the whole procedure to the participants at the beginning, they feel that everything is done so 'officially', which can cause a certain perception of pressure. As we all know, people can behave differently under pressure, either positively or negatively. For instance, we saw that almost every participant clicked on the 'Help' button and read all information carefully, whereas we know (not only from personal experience) that users tend to scroll through such manuals really quickly. In order to have a view that reflects reality as accurately as possible, we might have to track users' behavior when they are comfortably in front of their home computers, without them knowing. It goes without saying, however, that this method would have been highly unethical.

Chapter 6

Evaluation of functional modules

6.1 Continuous Integration

Although maintainable code can be written and reviewed by peers without running the code, there has to be a way to simply evaluate the functionality of newly written code, as well as making sure that the written code works. In order to achieve this, Tribler already has a continuous integration system in place: Jenkins[1]. In Jenkins, Tribler developers have created multiple so called jobs over the years which each run tests written for a certain part of the system on multiple operating systems, including MacOS, Linux and Windows. These tasks are triggered whenever a commit is made in a Pull Request on the Tribler repository. In order to test our own added modules in our separate repository, we have cloned the current jobs used in the Tribler repository in order to maintain the same quality standards as Tribler supplied. This in turn lead to a greater amount of tests being written, as well as a extra notion of code quality among our added code, which was checked as well by the continuous integration.
One hurdle we came across during our sprints was that we had added JavaScript tests, which initially were not taken into consideration by the continuous integration, leading to falsely decreases in code coverage, since the JavaScript code was not covered by the tests ran by continuous integration. Because of this, we added a new job to Jenkins, which made sure to run all JavaScript test, as well as report the covered lines which could in turn be used to calculate the true code coverage.
Concluding, our testing suite was mostly covered by the continuous integration in place for the Tribler repository, but an additional testing suite was added to accompany the JavaScript code.

[1] http://jenkins.tribler.org/

Chapter 7

Further development of the product

7.1 Actual Trust

Unfortunately, the notion of trust is a vague one. When do we consider a given user trusted? At first thought, one could say that someone is trusted if he or she will always be involved in and contribute to the network. But how do we know this? Can we base trust rating on findings from the past, of which we know that they give no guarantee for the future? And if we do, how can we be sure that nobody can ever find a way to trick the trust system, thereby undermining the network? Sadly, there is no clear answer to all these questions. When trying to invent a trust metric, there is the eternal fight between metrics that can straightforwardly be computed on the one hand, and metrics that give a more accurate indication on the other hand. A question that naturally arises is the following: does a metric that gives an accurate reflection of trust actually exist? Since we do not even know the answer to that question, we chose for inventing a metric that can straightforwardly be computed. However, we implemented this in such a way that the metric can really simply be changed in the future. As the discussion about trust will continue to develop, our product will keep evolving. Until the day that we have found a trustworthy trust metric.

7.2 Relevant nodes

The decision on which nodes to show is a complex one. If each user has traffic with one hundred others (which is a conservative number if you 'only' download 20 GB), showing two levels of neighbors will generate about ten thousand nodes. Limiting this to a number under one hundred can be done in many different ways. Our approach of only picking the connections with the highest amount of total traffic is one solution, but many others can be created. Finding a solution to limit the amount of nodes on screen without deforming the view of the network can be an entire context project on its own, because many factors can be used for filtering. Some examples are filtering on the total amount of traffic of a node, total amount of traffic to another node or the average amount of traffic of a node to all its neighboring nodes.

7.3 Alternate views

We created a view where the user can get an overview of the major nodes in the network around one user, but this does not give a clear overall view of the entire network. Also, detecting occurrences of Sybil attacks is really hard since the clusters created by this kind of attack are usually linked to the rest of the network through only one node, so finding a cluster requires a user to find the bridging node. To give a clearer overview of the network, a separate view could be created, displaying the entire known network. This view could also incorporate clustering algorithms to find parts of the network that are isolated and thus indicate Sybil attacks, making it easier for users to understand what Sybil attacks are and help the community find users who perform this kind of attack and exclude them from the network.

Appendix A

Evaluating Usability and Awareness

A.1 Debriefing questions

- Usability
  - Did you notice you can click on a circle? What happens when you do so?
  - Did you notice you can hover over circles and lines? What happens when you do so?
  - Did you notice the 'Back To You' button? What happens when you click it?
  - Did you notice the 'Help' button? What happens when you click it?
  - Did you like using this application? Please explain why.
- Awareness
  - Describe what you have just seen.
  - What does a circle represent, what does a circles text, size and color mean?
  - Is it clear you are a 'part' of the network, how is this visualized?
  - How can you influence the color of your own circle?
  - How are the colors of the other circles influenced?
  - What does a line represent, what does a lines size and divider mean?

A.2 User feedback

A.2.1 Feedback gathered in the first round

The main feedback and comments are paraphrased and summarized below.

- Tip
  - "What is PageRank"?
  - Missed both legend and question mark buttons.

  - "Put a delay on highlighting of the lines."
  - Not clear what the reputation means.
  - Users ask for explanation about meaning of circle text.

- Top
  - Network structure clear.
  - Circles are users/computers.
  - Lines are connections.
  - Notion there is some kind of reputation.
  - Beautiful animation, fun to use.

Adjustments

- Merge legend and question mark button into 'Help' button showing a help page.
- Replace PageRank reputation metric by easy to understand Balance metric.
- Delay on highlighting lines when hovering over lines.
- Use easy to understand and non technical naming.

A.2.2 Feedback gathered in the second round

The main feedback and comments are paraphrased and summarized below.

- Tip
  - Not clear which edge is highlighted when edges are overlapping.
  - "Why do I still see free riders? Are they blocked already?"
  - User tries right mouse click, this shows an unintended menu.
  - "How much can I download before I am blocked?"
  - "'Back To You' is animated in steps?"
  - Animation too fast, too much circles, hard to follow.

- Top
  - Network structure clear.
  - Balance and how to influence this is clear.
  - 'You' circle distinguishable from other circles.
  - 'Help' and 'Back To You' button found and clicked by most users.
  - 'Back To You' animation looks awesome.
  - "Fancy animation."

Adjustments

- Disable right mouse click.
- Limit the amount of circles (and thereby the amount of edges) shown.
- Add text quantifying when users will be blocked.
- Rename 'Trust Display' to 'Network Explorer'.