Secure Content Dissemination in Opportunistic Networks

Master Thesis

Carlos Anastasiades
Sascha Trifunović

April 24, 2009

Advisors: **Bernhard Distl, Dr. Franck Legendre**
Supervisor: **Prof. Dr. Bernhard Plattner**

Computer Engineering and Networks Laboratory, ETH Zurich

Abstract

PodNet is a distributed content dissemination architecture which allows exchanging content in an opportunistic way between mobile devices using ad hoc radio communications. Currently, participants cannot be held accountable for content they publish or relay since they are anonymous and not restricted in any way. PodNet is hence an ideal target for spammers and the dissemination of objectionable content. This thesis extends the existing PodNet design with integrated mechanisms, which provide the trusted and spam free distribution of user-generated content. We focus on the challenging case of a pure opportunistic network without the support of an infrastructure (e.g. gateways) or a central authority (e.g. PKI). We shift from a data-centric approach to a user-centric approach. The identification of participants is based on self-created credentials. Authors' reputation is assessed by a two level rating which measures legitimacy (objective rating) and quality (subjective rating) of the published content. The level of trust among users relies on the social ties between participants classified in categories, which are defined either consciously through secure pairing (e.g. friends) or inferred from a community detection algorithm (e.g. community members, familiar strangers, strangers). We show through simulations using real world traces and synthetic models that exchanging reputation information among users outperforms individual reputation collecting (i.e. no exchange) by reducing the spreading of unwanted content by at least 50%. The impact of liars (i.e. participants who deliberately spread false reputation) is reduced by weighting received reputations according to the source's social category. In addition, a spam control mechanism enslaves an author's publication rate to his/her reputation. It prevents unknown users from flooding the network and allows reputable users to publish freely. It moreover serves as an incentive for content consumers to rate an author in order to get more or less content from him/her. All the proposed extensions are implemented and successfully tested on both Windows Mobile 2003 and Windows Mobile 6 running on HP iPAQs and HTC Touch. The computational as well as the communication overhead is evaluated and compared to the existing unsecured implementation. Additionally, an existing community detection algorithm was improved and enhanced by a dynamic aging mechanism. The modified algorithm was evaluated using real world traces and produced better-connected communities in a shorter period of time. Overall, our schemes are not only relevant to PodNet but to opportunistic networks in general where reputation and trust must be enforced in a distributed, unsecure, and delay-tolerant mobile environment.

Acknowledgments

We would like to thank our two advisors Dr. Franck Legendre and Bernhard Distl for their great support and guidance during the course of this thesis. We appreciated their confidence allowing us to work very freely within the topic. They always had time for informative discussions, challenged our ideas and provided us very helpful tips which pointed us in the right direction. In particular, we want to thank Bernhard for all his constructive feedback on the security part which often helped us see problems from a different perspective. Special thanks also to Franck who provided us with very valuable know how in the area of social mobility and simulation models and who helped us greatly with the structuring of our report. His remarks and scientific advices were very valuable and helped increasing the quality of this report significantly. It was a pleasure to work with them.

Finally, we would like to thank Prof. Dr. Bernhard Plattner for giving us the opportunity to write our master thesis in this interesting area at the Communication Systems Group and for the supervision of our work.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

Due to the rapid proliferation of portable mobile devices, podcasting has become increasingly popular in a wide area of fields such as education, entertainment, news propagation, politics and marketing.
Traditional podcasting services [1] require servers across the Internet where producers of content can provide such, usually organized as episodes in specific channels. On the other hand, end users can subscribe to these channels in order to automatically fetch new episodes, whenever connected. The playback of content is often done offline when the user is no longer connected to the Internet.

The PodNet[1] project targets the mobile content distribution by extending the traditional podcasting concept for public (i.e. open and unrestricted) peer-to-peer content delivery among users. The current hybrid architecture is shown in Figure 1.1. Besides the traditional way of getting Internet content from fixed PodNet gateways, it consists of an opportunistic part which allows users to fetch new content (either from the Internet or user-generated) from other mobile users that are in proximity and carry the desired content.
In contrast to the fixed infrastructure, opportunistic connections lack any central authority and continuous network reachability. Since the users may be in motion, very short and unstable connections are possible and it is thus quite hard or even impossible to predict future encounters. In order to avoid any assumptions on reachability, all the data transfers are performed over one hop exclusively. Routing is replaced by the mobility of the users, since the user distributes the content while moving around instead of the content being distributed over several hops.

Content which is generated in a channel will be immediately delivered to all connecting and subscribed users. After reception, those users will supply it to all other subscribed users which connect to them. Thus, there is a clear difference between *author* and *supplier* of content in PodNet. In contrast to authors which have produced the content, suppliers cannot be held accountable for it because every content is automatically supplied to others after the download, even without reviewing it.
Currently, content publication is done anonymously without any restrictions. There is no notion of a user's identity which could help to identify the content's authors, nor is it possible to check the integrity of content. This makes PodNet an easy attack target since it enables spammers to distribute a huge amount of spam messages for free and allows fakes to spread rumors and criminals to exchange illegal content, all without having to fear any consequences as no easy

[1]http://www.podnet.ee.ethz.ch

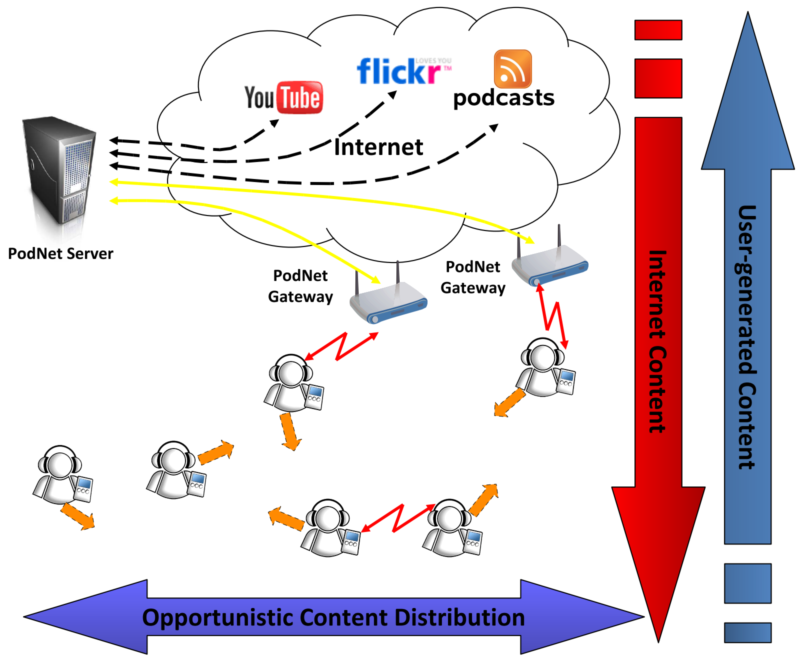


Figure 1.1.: PodNet Architecture.

identification is possible.
In order to hold users responsible for what they publish, a proper user definition including the associated authentication is needed.

1.1. Tasks

Since PodNet was designed to allow mobile devices to exchange podcasts in a completely distrusted way, it is important that a security design does not necessarily need any central infrastructure. Although the architecture of PodNet is hybrid providing fixed gateways connected to servers, we constrained ourselves to the pure opportunistic part of PodNet. However, assuming the central authority being an optional add-on, it could help verifying the authenticity of real users.

This thesis comprises two main tasks, namely the trusted content exchange based on user reputation and the spam free content publication in the opportunistic environment of PodNet. As a main requirement to fulfill both tasks, user identities need to be introduced in order to bind published content securely to its authors and increase individual accountability. The detailed tasks can be found below.

Task 1: Securing both PodNet dissemination paths shown in Figure 1.1, i.e. the distribution of Internet content from a server and the distribution of user-generated content. The individual accountability of user-generated content should be increased by author ratings which are exchanged if possible before downloading the content. The decision whether to trust received information or content should rely on the social ties between peers.

Task 2: The study of classical cryptographic solutions against spam and finding proactive solutions which regulate publication in order to prevent spammers from flooding the network. The proposed solution should solve the problem either by social regulation or by introducing a rate limitation.

1.2. Problem Formulation

Several issues arise when trying to approach the tasks specified in Section 1.1. They can be mainly divided into the following topics: identity, distribution, reputation, accountability, rating and tradeoff.

Identity: In order to achieve individual accountability and exchange reputation information, user identification and authentication is needed. Therefore, it is crucial that unique inimitable user identities are generated. Using a centralized authority which generates the unique identity at an initial registration process would solve the problem but would require an infrastructure which is not available in opportunistic networks. Allowing every user to generate and declare his or her own identity would enable the usage in a completely distributed environment but ensuring the uniqueness of identities becomes more difficult and avoiding the creation of multiple identities infeasible.

Distribution: The security measures should prevent bad content from spreading but allow legitimate content to spread without restriction. Considering the PodNet distribution in Figure 1.1, the distribution of user-generated content becomes the most challenging of all distribution schemes because the authorship is not limited to any well known group and everybody, even unknown users, can create content. The question arises, whether to trust an unknown author and to allow the content download or not.

Reputation: The main problem that a reputation system in PodNet faces is the fact that the network is opportunistic which means that the users are not continuously connected to each other like in the Internet. Requesting information from a centralized database or other users on demand becomes infeasible because on the one hand, no other users or servers may be in range, and on the other hand, the decision whether to download content should be made fast since users may move away as well. Every user has only an incomplete view of the network and therefore, as a consequence, has to rely on other users' opinions (see *Rating* below) about an author. However relying on information of others increases the susceptibility to liars and the question arises whether to trust a single user's information if no other ratings are available.

Accountability: In most cases, it is not possible to hold suppliers responsible for the downloaded content because in a PodNet environment, suppliers and authors may not be the same. In particular, when subscribing to a channel, one would automatically download content which is included in this channel from other users and at a later stage, supply it to others (become a supplier) without necessarily having reviewed the content yet.

Rating: One way to give an opinion about other users as well as content is through rating. But what information a rating should disclose is arguable. A rating can be considered from two different angles. One one hand, one could rate the legitimacy of the content whether it conforms with the description (objective rating) and on the other hand one could rate the user satisfaction with the content (subjective rating). Assessing the user's satisfaction is difficult because every user may be different in behavior and taste. The difficulty in generating a consistent subjective rating arises in particular because a user's taste may influence his/her perception of the content quality as well as the priority given to it.
When allowing multi-level rating, a more detailed rating is possible but a user that always rates near the maximum values will gain a higher influence than several other users that rate smaller values. Besides that, ratings may be depending on their context, i.e. trust in a user's opinion about content related to sports does not necessary lead to an agreement in the taste of music. Additionally, ratings may evolve over time.

Tradeoff: A reasonable tradeoff between security, performance and usability has to be found as explained in Section 3.1. From a security point of view, one would want to exchange all possible information with all other users in order to get a nearly complete view of the network but unfortunately, resources are limited on mobile devices and the sending overhead should be reduced to a minimum in order to use most resources for the actual data transfer. The dissemination of malicious content should thus be prevented but without affecting the exchange of legitimate data. The user should be able to assess the quality of received content and regulate the data exchange by that, but at a level of abstraction not confusing him or her.

1.3. Summary

This thesis comprises the extension of the existing PodNet by a security concept which allows for a secure and spam free content distribution. Currently, participants are anonymous and not restricted in any way when publishing content, which allows for an easy distribution of spam and

objectionable material. Although PodNet is a hybrid architecture, we only focus on the pure opportunistic content dissemination between mobile user, disregarding any possible infrastructure such as central authorities.
In order to achieve this goal, one has to be able to hold users accountable for what they publish. For this reason, a proper user authentication was introduce based on self created credentials. This guarantees the inimitability of an identity since every user can be challenged for the private key corresponding to his/her credentials. Preventing users from generating multiple identities and launch sybil attacks is not that easy. In order to minimize their effect, every identity is assigned a trust value which reflects the believe of an identity being genuine.
The level of trust between users depends on their social ties which are classified into different categories. The most trusted users are *friends*, which are obtained by a secure pairing process. By exchanging friends list a *friend circle* containing 2-hop friends is built, whose trust is still high. Other trust levels are achieved by a community detection algorithm which was improved and enhanced by an aging mechanism. The algorithm classifies surrounding users into *familiar strangers* and *community members* which are each assigned a certain trust value. All other users are treated as strangers, having minimum trust.

Content can be generated and shared for many different purposes in PodNet. For this reason our design introduces three channels, namely *open*, *restricted* and *closed*. Traditional podcasts may be published in *restricted* channels where only a few people are authorized to publish but everybody may view content. *Closed* channels are encrypted, guaranteeing privacy for a small group of people, e.g. friends, that want to share photos. The most challenging distribution takes place in the *open* channel where everyone is free to publish.
In order to guarantee usability, users can now rate authors of content and asses their reputation by sharing the acquired opinion with others. A two level rating was introduced which measures legitimacy (*objective rating*) and satisfaction (*subjective rating*) of the published content. Authors of illegitimate content are blocked (i.e. put on a blacklist) and this information is shared among users. In case enough recommendations from other users are received, the author is blocked as well. The impact of liars trying to manipulate the reputation is reduced by weighting the received recommendations according to the source's social category.
The *subjective rating* is used to regulate the publication rate of an author. This serves as a proactive spam control mechanism which avoids the flooding of unwanted content and at the same time allows reputable authors to publish freely. It additionally serves as an incentive to rate an author, in order to get more or less content from him/her.

For the purpose of assessing the effectiveness of the proposed design, the different security measures were simulated using real world traces and synthetic models. We showed that exchanging reputation information among users outperforms individual reputation collecting (i.e. no exchange) by reducing the spreading of unwanted content by at least 50% depending on the data set. Additionally, we assessed that by applying social based weights to the recommendations we could reduce the influence of strangers by 50% and still achieve a higher detection speed of unwanted content.

The security concept was integrated into the existing PodNet implementation and successfully tested on both Windows Mobile 2003 and Windows Mobile 6 running on HP iPAQs and HTC Touch. The overhead in communication and computation of the Secure PodNet was evaluated and compared to its unsecured predecessor. Additionally, the performance of the enhanced community detection algorithm was evaluated using real world traces. Thanks to its modification it produced better-connected communities and through the dynamic aging mechanism the detection would take a much shorter amount of time.

All in all, our schemes are not only relevant to PodNet but to opportunistic networks in general where reputation and trust must be enforced in a distributed, unsecure, and delay-tolerant mobile environment.

1.4. Contributions

The contributions of this thesis comprise the design, simulation, implementation, and evaluation of security mechanisms that ensure the trusted and spam free content exchange within PodNet. The main contributions consist of the following points.

- Design of a distributed *identity management* allowing the creation of unique IDs which can be authenticated for different authorization profiles depending on the channel class namely open, restricted, and closed.
- Definition of *trust metrics* relaying on social ties such as friends (established through secure pairing) or community members (inferred from an enhanced community detection algorithm).
- Appliance of a *reputation system* based on a two-level rating which allows to assess the legitimacy (objective rating) and the satisfaction (subjective rating) of content and share this information with surrounding users.
- Introduction of a *spam control* mechanism consisting of shared blacklists (reactive) and a reputation dependent rate limitation (proactive).
- Simulation of the designed security measures using different traces and synthetic models in order to assess their effectiveness.
- Implementation of the Secure PodNet and evaluation of its overhead in communication and computation as well as the quality of the community detection algorithm and the benefit of the added aging mechanism.

1.5. Outline

The remainder of this report is structured as follows: Chapter 2 presents background and overview of state-of-the art solutions; Chapter 3 describes system requirements and design. Simulation results are shown in Chapter 4. The implementation is explained in Chapter 5 and evaluated in Chapter 6. Chapter 7 concludes our work and gives prospects for future investigations.

2. Related Work

This chapter gives a review about the scientific literature of related work. The extension of the traditional podcast concept to a hybrid PodNet is described. Additionally, the distribution of certificates for classical as well as for self-organized systems is discussed. Thereafter, the different definitions of reputation and trust are presented and how trust can be propagated. Furthermore, some approaches to counteract sybil attacks are summarized. Finally, some existing community detection algorithms are described.

2.1. PodNet

In [2], the traditional podcast concept is extended for public (i.e. open and unrestricted) peer-to-peer content delivery among mobile users. It consist of a hybrid network where peers can share content among themselves in an opportunistic way or download it from the Internet through the PodNet gateways. Content is organized as episodes in channels and users solicit episodes in subscribed channels. Every participant is able to create new channels and generate content anonymously and without any restriction. Once content is published it is distributed automatically to anybody who is interested in it.
In order to advertise an existing channel, it is added to a predefined discovery channel which every node carries and updates automatically. This channel contains information about all known channels of a node. If a user is unsure about what channels to ask for, he/she can consult the contents of the discovery channel and select (i.e. subscribe to) all he/she is interested in.
The implementation of the concept in [3] provides synchronization and basic content distribution functionality. The distribution is data-centric and does not provide any notion of a user. Every user periodically broadcasts discovery messages in order to inform other devices about its presence. After randomly connecting to one of the present peers, synchronizing the common channels and selecting all desired episodes, the data is exchanged in a way similar to BitTorrent [4].

2.2. Certificate Distribution

In a classical certificate architecture as described in [5] only an certificate authority may hand out certificates to end user. These certificates are signed by the authority in order to be valid. For a user to be able to check whether a certificate is valid he/she needs to hold at least one certificate of a trusted certificate authority which was previously acquired through a secure side channel or a setup process, as for example with the installation of a web browser. With the trusted authority's certificate the other user's certificate can be verified. For the purpose of dividing the load, it should be possible for more than one authority to exist simultaneously. The authorities are linked in a hierarchy which allows a user to verify certificates signed by authorities not known directly.

A self-organizing public-key management system is proposed in [6]. Similar to PGP [7], key authentication is performed via chains of public-key certificates. Every user creates its own certificate and exchanges it over a secure side channel with trusted devices. These devices can then issue the public-key certificate by signing and returning it to the owner if they believe in its correctness.
In an initial state, every user only holds his/her own certificate signed by other devices and the certificates he/she signed for others. Every node periodically polls its physical neighboring nodes for unknown certificates with which certificate chains can be built. When two nodes meet, they merge their repositories and try to find appropriate certificate chains.
The advantage of such a system is the fact that it reflects social relationships. However, in order to verify a final certificate, all intermediate certificates in the chain have to be known and valid. If one certificate is unknown or has expired, the chain breaks. Therefore, it is assumed that all nodes can establish communication with any other certificate issuer within validity period. They claim in [6] that any user would find at least one certificate chain to any other user in their merged repository with a high probability but this approach assumes that all the peers have well established local repositories, are socially well connected and have consciously signed many social relationships.
The establishment of security associations among users (e.g. for issuing certificates) is considered in [8] in a more general way. Two general approaches are presented: one being fully distributed and the other using an off-line central authority. The central authority uniquely binds a node's identity to the public key, allowing automatic certificate exchange when nodes are in radio range but needs additional infrastructure at initialization. In the fully distributed approach, friends consciously exchange their certificates either over a secure side channel (as in [6]) including a challenge response scheme or through a common friend. The latter approach cannot be automated and leads to problems when meeting complete strangers since in contrast to [6] no certificate chains are build because trust is assumed to be non-transitive. The identification via common friend is assumed to remove any ambiguity because he/she knows only a limited amount of people. In both [6] and [8], the issuing entity checks whether the public key identity binding is unique and correct.

2.3. Reputation and Trust

Reputation and trust are often confused and used interchangeably in literature although they do not mean exactly the same. Trust towards a user defines the believe that a user will show a certain behavior in the future whereas reputation is based on past experiences and observations

regarding the users' behavior norms [9]. However, most works only consider one of both topics. Due to the huge amount of different research work, we are presenting only a small selection of it.
Pretty Good Privacy (PGP) [7] proposes a decentralized authentication mechanism based on a *Web of Trust*. Users can issue the trustworthiness of others' public key bindings and validate the trust in unknown users based on chains of aggregated trust relations. Trust is assumed to be completely transitive and all the relations are subjectively assigned. The approach is static since the trust values cannot change over time with the user's behavior.
A more dynamic trust model is presented in [10]. The model incorporates both direct trust based on previous experiences and recommended trust values received from other users. False recommendations are adjusted by applying the semantic distance between the recommended and experienced values to obtain a corrected value. However, the corrected value will still be inaccurate if the difference varies a lot.
A bayesian formalization for distributed binary ratings is proposed in [11]. The system differentiates between complete strangers which are ignored because no information is available and known strangers which were never met but known from other users recommendations. If not met before, trust is calculated recursively via known users and chains of trust.
A reputation system based on a modified bayesian approach is proposed in [12], [13], [14] and [15]. In this approach the past experiences are weighted with a discount factor which serves as fading mechanism. Ratings received from others are considered with less weight in order to trust the own observations to a higher extent. Fake ratings are removed by applying a deviation test which neglects ratings that differ by more than a threshold value from the other ratings. Such a deviation test successfully removes ratings from extreme liars but does not detect strategic lies below the threshold value which adaptively change a user's rating towards the wrong state. Additionally, the behavior test introduces a delay in detecting the correct reputation value after a user's behavior change.
In contrast to previous work, the system in [9] separates reputation for providing service/content from reputation for providing recommendations and thus giving a sophisticated and accurate view on reputations. The system represents its reputation value in the continuous range between -1 and 1. Intermediate values may define the degree of trust, as above 0.2 means trustworthy and above 0.8 very trustworthy[1]. The actual reputation value is a combination of the old and new reputation value using a fading factor which regulates the importance of new aggregated values and evolves the reputation over time. For all recommendations, the difference of recommended value and direct observed behavior is computed dynamically and a maximum deviation is allowed. Obviously, recommenders that deviate less from their own experience are given more weight. Recommenders exceeding the threshold are ignored even though they are not necessarily malicious but may have a different taste.
As incentive to rate content properly, reputation information with others is only performed with peers that have good recommendation reputation. In the combination process of the reputation values, the most weight is given to one's own observations followed by recommendations from

[1]Symmetric for negative values and distrust.

recommenders which are very similar to one's direct experiences.
Finally, in [16] a bayesian trust framework is proposed which extends the model in [9] by allowing n discrete trust levels. Assuming $n = 4$ different rating levels, the initial trust tuple d for another user is uniformly distributed, i.e. $d = (0.25, 0.25, 0.25, 0.25)$, for equal probability for all four rating values. For every new rating, the value at the position corresponding to the rating value is increased. The variance within the trust tuple indicates the confidence in a trust value, i.e. if the variance is high, a user has high confidence in a value whereas when being zero (uniformly distributed), the trust is minimum. A similar trust tuple is build for every user recommending another peer. The values of both, direct and recommended trust, are then combined using a weighted sum where the weighting is based on the confidence level of direct over recommended trust as well as the subjective reliance on one's own personal ratings.

2.4. Trust Propagation

In the previous section, we considered rating and reputation calculations. Here, we look at the propagation of reputation and trust. In all of these approaches, each user sends his/her rating values to other users which calculate a reputation value based on their own direct observations as well as all received indirect observations. However, these approaches require quite a large overhead since not all ratings may be needed. In [17] and [18] a method is presented which is well suited for opportunistic networks because of its lower sending overhead. When a rating is made, the rater sends the rating back to the rated user which can present it to other users when they want to assess his/her trustworthiness.
The model neglects the problem of certificate distribution completely and assumes that every device has a unique identifier.

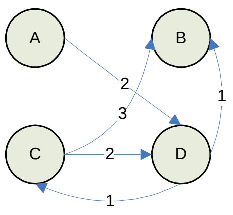


Figure 2.1.: A Simple Web of Trust Example: *Taken from [17].*

The lightweight trust distribution algorithm proposed in [17] allows every user to calculate a subgraph of the web of trust. In particular, each user stores both the ratings generated for other users (outgoing ratings) and all ratings from users who have rated the user itself (incoming ratings). Whenever two users meet, they will exchange their incoming ratings and store the other user's ratings in order to build a subgraph of the web of trust. When device A in Figure 2.1 meets an unknown device B, it would calculate a trust value by first determining the performing relation[2] and the judging relation[3]. In order to be able to calculate both these relations, all the devices in the subgraph should be well connected and have rated each other[4]. The trust value is calculated after applying semi-supervised learning techniques on the most related relationships. The method is inherently resistant to sybil attacks[19] since sybil users are not connected the same way in the web of trust as regular users and therefore do not influence the trust propagation. However, we will look at sybil attacks in Section 2.5.

When using this mechanism, it is crucial that the users cannot modify their own ratings locally and therefore, a tamper proof storing mechanism is introduced in [18] which forces user to store received ratings and prevents them from changing them or inventing ratings without having received anything. The key idea is that a recursive hash chain is calculated over all the local ratings always including the previous hash value and the new rating value in order to prevent local tampering with the rating.

When a device A wants to rate a device B, it stores the rating locally in its own hash table before sending it to B which also stores it in its hashing table. After having stored the rating, both devices send a commitment message comprising the rating and hash entry to a witness node[5] which should check the integrity and authenticity of both rating tables and therefore ensures that both have correctly stored the rating. Despite the problem of agreeing on good witness nodes and finding reasonable timeout values for the commitment messages, the mechanism may work well if the supplier and author of the content either coincide or meet each other regularly so that they can determine witnesses and exchange the locally stored ratings. However, if author and supplier are not the same person and the downloading node would never meet the author, the rating exchange and witnesses selection would be very difficult. It may be reasonable that in such a case, the downloader would not store and exchange any rating at all since it would not be clear where to send these commitment messages but this would mean that only devices meeting each other regularly could rate each other.

In order to store rating tables from growing over time, ratings of the same relationship could be summarized or discarded after an expiration time which is all not further elaborated in [17].

2.5. Sybil Attacks

A distributed environment lacks a central trusted authority which could guarantee one-to-one correspondence between entity and identity. Therefore, an attacker can try to distort a reputation system by creating a large number of identities for the same entity in order to gain a larger influence.

Alternatively, an attacker may also profit from creating new identities to leave the bad reputation of the old identity behind and being able to misbehave again.

[2]Relation between devices that are rated by the same rater A (itself).

[3]Relation between devices that have rated the same device B (interested user).

[4]In the example above, device C should have had contact with both D and B. This assumes that all rated devices have provided content to the others.

[5]It is not specified how to select witness nodes but it should be in proximity of the device being rated (i.e. B) in order to replay the message to device B if it was temporary unavailable.

The difficulty in the PodNet system is not only the distributed environment but also the fact that the network comprises mobile nodes which communicate in an opportunistic way. There are different angles on how to tackle this problem. In [20], it is assumed that all identities of a sybil attacker are bound to a single physical node and move together. In contrast to a group of friends communicating simultaneously in parallel, sybil users can only communicate serially and could thus be detected because they generate much fewer collisions on the MAC layer. The disadvantage of this approach is the fact that it does not detect sybil users which are not simultaneously used as well as fast switching between identities to quit bad reputation.

Other approaches consider the social network of individual users. SybilGuard[21] assumes that malicious users can create many identities but only few trust relationships and honest nodes have good connectivity to the rest of the network. By determining whether the graph has a small quotient cut[6], it can be estimated whether nodes are sybil identities or not. They claim that social networks do not have such edges. However, they approach the problem for a routing based environment observing the randomly selected routes of the individual nodes and stating that routes from all sybil nodes always lead through the same edge.

Although not directly applicable to PodNet because it is based on one hop communication only, exploiting the social relationship may be a valid concept. Similar to the lightweight trust distribution algorithm in Section 2.4, one can assume that sybil nodes are not connected as well as honest users.

In [22] a distributed trust model is proposed that uses anonymous authentication based on blinded threshold signatures. They claim that a user wants to be anonymous in the system to keep his/her privacy by using different pseudonyms, inside different communities. They refer to [23] in which it is concluded, after formally analyzing with game-theoretic models, that when allowing users to freely change identities, all unknown users should be regarded as malicious. As a solution not to mistreat all unknown actors, they proposed to use a free but not replaceable (once in a lifetime) pseudonym which is certified by a central authority.

The user reveals his/her certificate including his/her (once in a lifetime) identity to the other community members which should know the identity in order to verify that no other member is using this identity. In the following, the requesting user generates new credentials (private/public key) which are used as pseudonym for that community and let the community members sign the certificate using threshold cryptography. Although they claim that the method is fully distributed, it still needs an initial (once in a lifetime) identity and it is not further specified how community members verify that the same user did not already joined the community with another identity.

[6]Small set of edges whose removal disconnects a large number of nodes (sybil identities) from the rest of the graph.

2.6. Community Detection

Community detection can be used in order to automatically detect the importance of a peer encountered regularly. It classifies users into different categories such as familiars (people encountered regularly), community members (people closely linked by a lot of common familiars) and possibly others, depending on the algorithm used. All other users are treated as strangers which means they are not known or not seen regularly. These classifications can be used to establish trust, i.e. we could give an opinion of a community member more importance while disregarding strangers completely.
Unfortunately, to the best of our knowledge, not much work is available in the area of real-time distributed community detection in delay-tolerant networks. One of the few published works is [24] in which three community detection algorithms are proposed, namely *Simple*, *k-Clique* and *Modularity*. They differ in level of computational complexity and resource management.

Algorithm 1 Simple

```
$f\_th_{add}$: Familiar set adding threshold
$\lambda$: Adding ratio
$\gamma$: Merging ratio
$n_i$: A node
$F_i$: Set containing all familiars of node i
$C_i$: Set containing all community members of node i
$n_0$: Local node
$F_0 = \emptyset$
$C_0 = n_0$
for all nodes $n_i$ in proximity do
    acquire $F_i$ and $C_i$ from $n_i$
    update $n_i's$ contact time $t_i$
    if $t_i > f\_th_{add}$ then
        add $n_i$ to $F_0$
        if $\mid F_i \cap C_0 \mid / \mid F_i \mid > \lambda$ then
            add $n_i$ to $C_0$
            if $\mid C_0 \cap C_i \mid > \gamma \cdot \mid C_0 \cup C_i \mid$ then
                merge $C_0$ and $C_i$
            end if
        end if
    end if
end for
```

Algorithm 1: Simple

The *Simple* community detection mechanism is described in Algorithm 1. It requires three design parameters, the familiar set adding threshold f_th_{add}, the adding ratio λ and the merging ratio γ. Once the total contact duration of a neighbor exceeds f_th_{add} the corresponding neighbor n_i is added to one's familiar set F_0. In case the adding criterion

$$\mid F_i \cap C_0 \mid / \mid F_i \mid > \lambda \tag{2.1}$$

holds, n_i is also added to one's community set C_0. Additionally, the aggressive version of the algorithm merges the two communities while the merging criterion

$$| C_0 \cap C_i | > \gamma \cdot | C_0 \cup C_i | \tag{2.2}$$

is true.
In summary, as the name already implies, the *Simple* algorithm requires the fewest overhead in computation, storage and sending, but the merging of similar communities in order to speed up the detection decreases the accuracy. Besides that, three design parameters have to be set carefully because they may change in different environments.

Algorithm 2: k-Clique
The *k-Clique* algorithm is very similar to *Simple* but the adding criterion is the following:

$$| F_i \cap C_0 | \geq k - 1 \tag{2.3}$$

This algorithm needs more data storage and sending overhead since the familiars of every community member are exchanged and the decision whether to include a node from the other user's community is made individually by checking Equation 2.3 for the corresponding community member.

Algorithm 3: Modularity
Similarly, *Modularity* needs more storage and sending overhead, but additionally to *k-Clique*, the calculation overhead is increased since the modularity difference has to be re-calculated for all nodes whenever a node is added to the community.

All three proposed algorithms suffer the following disadvantages which are all highly important in the PodNet environment:

- The performance evaluation of community detection in [24] is only considered after replaying the entire traces and not adaptively during the replay process.
- Different design parameters are used depending on the environment in order to achieve the best results in the processed traces, e.g. for the familiar set adding threshold f_th_{add}.
- The algorithm does not consider aging of contacts, thus ignoring that contacts lose their importance after a large time of inactivity, as well as the uncontrolled growth of the community size it leads to.

3. Design

In this chapter the design of the Secure PodNet is illustrated in detail. At first, the different requirements and the attacker models are presented. Then, the main design aspects consisting of the authentication and the authorization of users are described. Furthermore, the trust metrics used, as well as the design of the reputation system are shown. Finally, the measures taken against spam propagation are pointed out.

3.1. Requirements

The main requirements of PodNet can basically be divided into three parts as shown in Figure 3.1: *security*, *usability*, and *performance*. It may not be possible to meet all requirements entirely and therefore a tradeoff between security-usability and performance has to be found.

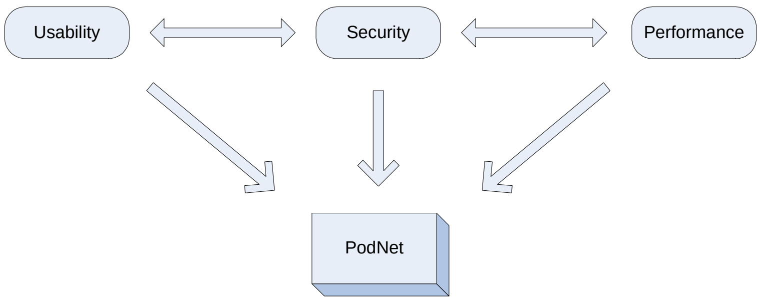


Figure 3.1.: PodNet Requirements.

Security Requirements: Security measures should improve the reliability and robustness of the content dissemination. By content reliability we mean the possibility to hold people accountable for what they do, either because they publish very good or bad content. Content from spammers should be prevented from spreading whereas good content should be distributed throughout the network. As a consequence, limiting bad content from spreading also avoids the depletion of resources.
The main goal is to be able to asses trust of a user even if there was no contact in the past. The user could be an author, who's information should be available even if no content was ever downloaded from him/her. This information could be obtained in advance from users (who's trustworthiness should also be known) which already have received content from that author, either directly or through multiple relay order. Depending on the trust in that user, which can

be based on the social relationship one has to him/her, the reputation information can be trusted or not.
Therefore, unique, non-imitable user identities are needed, so that no user can claim to be another existing user and content can be uniquely assigned to the user that published the content.
However, all security measures should not be too restrictive since PodNet's main goal still remains content dissemination in an opportunistic manner.

Usability Requirements: The system should allow the user to assess the legitimacy of and his or her satisfaction with the downloaded content. Ideally, information about content and author should already be provided in advance, i.e. before downloading the content. However, at the cost of accuracy, the rating and information display should be performed at a level of abstraction not confusing the user.
Furthermore, the system usability will be improved by requiring only minimal user interaction because too much interaction would provoke the user to turn off the system. All the processing should be done in the background without the user noting it. This means that the system also automatically decides whether to perform a certain operation or whether to download and trust content. Automatic decisions always bear the risk that attackers could try to fool the system rules.

Performance Requirements: A good performance is required in PodNet, especially since the computational power and energy on mobile devices is limited. Every usage of cryptographic functions, especially asymmetric cryptography, should be well reasoned and only applied if really needed. Therefore, the exchanged information should be limited to the minimum, e.g. only exchanging information about the most recent authors in common channels.
Nevertheless, in Security, we want to know and exchange information about authors with all communicating devices trying to obtain a complete picture of all authors. However, exchanging information about all potential authors would be a huge communication overhead.

3.2. Attacker Model

In this section possible attack scenarios can be found, since it is important to know the threats to a system before designing countermeasures against them. Therefore, we examine possibilities of how adversaries could attack a distributed content dissemination system such as PodNet and give a sketch of our countermeasures presented in the following sections. Depending on the attackers ability, the attacks can be classified into exploiting attacks, cheating attacks and low level attacks.

Exploiting Attack: This kind of attacker tries to exploits the system by properly using it. The attacker is not able to change the functionality of the program. Some examples are: positive and negative lying about rating, spamming or flooding as well as spreading of illegal or fake content.

Cheating Attack: A cheater is able to modify the functionality of the program. Some examples of this kind of attack are: sybil attack (many user IDs on the same device), ID and channel theft (impersonate person by using his/her user ID, reuse channel ID for completely malicious channel), no cooperation (freeriding), ignoring of restriction and rules (e.g. send rate limitations, ratios) and enclosure manipulation (sabotage complete file download by corrupting only one, not identifiable, enclosure).

Low Level Attack: Low level attacks are independent of the system itself but exploit the weaknesses of wireless communication. Some examples are: replay and wormhole attacks, IP and MAC address spoofing, man-in-the-middle attacks, jamming and DoS attacks.

3.2.1. Countermeasures

In this work, we add a proper user authentication and authorization to PodNet to counteract possible *cheating attacks* or at least minimize their influence. Additionally, we introduce the concept of highly trusted friends, distributed rating and blacklists as well as publication rate limitations in order to reduce the effect of *exploiting attacks*.
In order to generate the rating and blacklists, a two level rating which measures the legitimacy (*objective rating*) and satisfaction (*subjective rating*) of content is introduces in Section 3.6. Rating the content of an author will influence his or her reputation in the following way. If content is rated as objectively bad, the author will appear on a blacklist whereas if it is subjectively rated either good or bad, the author will appear on a rating list. The blacklisting mechanism comprises four different blacklists which are defined Table 3.1.

Type	**Content**
Personal Blacklist	personal opinion of the user itself (per user)
Local Blacklist	all received opinions received from one's surrounding (per user)
Channel Blacklist	misbehaving authors inside a channel, added by channel administrators (per channel)
Global Blacklist	misbehaving authors in several channels, generated by a central authority (global scope)

Table 3.1.: Overview of Blacklists.

Except for global blacklist, which has global scope, all author reputation values are only valid in a specific channel. The personal and local rating lists are defined likewise but comprise subjective ratings. In contrast to blacklist, no channel or global rating lists are defined.

Whenever security mechanism are designed, it is crucial to make sure the additional overhead does not introduce new attack points. Therefore, we have to ensure the rating and blacklists itself cannot be used as part of a flooding attack. This could be fulfilled for example by defining a maximum list size and always including only the most current entries into the corresponding list.

3.3. Authentication

In order to hold users responsible for what they publish, a proper user definition including the associated authentication is needed. The most crucial part comprises the generation of a unique, not imitable identity without requiring a central authority which regulates its selection. In this section, we will first focus on the identity generation and the required authentication mechanism in order to check another user's identity. Then, the signing process is explained, including how to establish special trust relationships between users. Finally, the credentials update process is presented.

3.3.1. Identity Generation

The system should work in a distributed environment allowing a user to create his/her own self-signed credentials without an initial registration at a central authority (CA). As presented in Section 2.5, in the absence of a central authority, it is possible for an attacker to generate several identities, i.e. sybil identities. We could require every device to initially register at a central authority in order to obtain a once-in-a-lifetime identity but this would limit applicability in a totally distributed environment. Therefore, we decided to have as an only requirement that a user cannot easily imitate or steal the identity of an existing user. We are aware of the fact that multiple identities are still possible but we will tackle that problem from a different angle by considering the social connections of a user, either through friends, the community[1] or recommendations[2] from trusted users.

The identity is built from the hash of a public key as done for the IP address described in [25], [26] and [27]. In order to successfully steal another user's identity, an impersonator would need to find the corresponding private key or at least find one that has a public key which hashes to the same identity. Both options are not an easy task.
Optionally the credentials can be signed at a later stage by a partially offline CA through a registration process and/or by friends through secure pairing [8]. Both mechanisms are shortly explained in Subsection 3.3.2.

3.3.1.1. User Credentials

The user credentials are shown in Table 3.2. The basic part of the credentials is generated at first program startup and contains user name, serial number, validity period, algorithm ID and

[1]See Section 3.5.
[2]See Subsection 3.6.2.

User Credentials	
Body	1. ID (Hash of Public Key) 2. Serial Number (random number) 3. Name (user input) 4. Validity Period 5. Algorithm Identifier 6. Public Key
Signature	Self-Signed Signature (over Body)
Additional Signatures	1. History: • previous ID • Signature (over Body) (with previous Private Key)
	2. Authority: • CA ID • Signature (over Body)
	3. Friends: • List of Friend IDs • List of Signatures (over Body)

Table 3.2.: The Layout of User Credentials.

public key. The name is a user prompted string which serves as a human readable identifier in PodNet. The randomly generated serial number additionally helps a user in differentiating between possible name collisions. However, the identity used by the protocol is still the hash of the public key. The validity period is set to two years[3] and the algorithm identifier specifies the algorithm used for the signatures.
The additional fields comprise the *history*, *authority* and *friends* fields. The *history* field is used in the update process (explained in Subsection 3.3.3) whenever a user renews his/her credentials. In such a case, the *history* field shows his/her old ID[4] which helps other users updating their internal reputation tables for this user.

3.3.1.2. Challenging Process

Since credentials with the copied identity and name can be forged, a peer has to prove his/her knowledge of the private key.
This challenging process will be performed in two cases:

1. At startup: If an unknown user is connecting, it will be challenged to check whether he/she really carries the private key. Additionally, it may be a measure against fake attacks. After generating a fake ID, an attacker would have to perform challenge request with all other users which is a computational overhead for him/her.

2. Before data download: A supplier of content has to authenticate itself before people will start downloading from him/her. By this means, the downloading peer knows that the

[3]See Subsection 3.3.3 for more information.
[4]Hash of the previous public key.

communicating peer is using his/her own identity. This is important for several reasons: If some data is corrupted or malicious, the information about the peer can be remembered and appropriate countermeasure can be taken in the future. Besides, there may exist some information which should only be exchanged if the communicating peers belong to a certain group of people, e.g. showing meta information of a *closed* channel[5].

This challenging process is flexible and does not necessarily need to be symmetric. It may need a challenge response or challenge request response depending on whether one or both peers need to authenticate themselves.

3.3.2. Signing Process

When allowing self-signed credentials, it is not possible to relay on the consistency of the credentials. Therefore, it is possible to additionally sign the credentials by either a central authority or friends in order to increase trustability. In the following two paragraphs we will briefly explain these processes.

3.3.2.1. Registration

In a registration process, a user connects to an authority which signs the users credentials after verifying its correctness. Signatures from central authorities (CA) ensure human identity since in contrast to self-signed credentials, signatures from CAs cannot be automatically generated. In order to make sure that a user can only generate one or at least a very limited number of CA signed credentials the issuing authority could demand a mobile phone number and require a SMS[6] confirmation. The number should not be included in the credentials but stored locally to prevent the same user from generating more than one identity. In this approach a problem arises if a user loses his/her private key, e.g. when accidentally reseting the device. This could be circumvented by not preventing a user to generate a new identity with the same phone number forever, but only for a limited time, e.g. a month.
Another solution would be to require an even more complex process in order to unblock a phone number, maybe even one that requires human interaction. One could even go one step further and perform the entire registration process per SMS or MMS[7] from a mobile device. This would make the registration process quite location independent in urban areas. Since PodNet is expected to work on mobile phones, the requirement for SMS reception may be an acceptable restriction.
Another option would be to require stronger authentication by registering social security or driver's license number which would raise the barrier for launching a sybil attack but the need for sensitive information may also scare away many potential users [21].
Authorities could advertise their services in the neighborhood using a flag in the service mask

[5]See Subsection 3.4.1 for more information.
[6]Short Message Service.
[7]Multimedia Messaging Service.

(see Paragraph *Services* below) of the discovery message[8]. Note however, that authorities are not yet included in the current implementation.

3.3.2.2. Secure Pairing

In this process, friends sign each others certificate as proof of their friendship and by that, increase their trust in each other.
Both users have to agree on a password and start a secure pairing process where authentication is performed through a fixed three way challenge request response mechanism. The authentication is done by encrypting one's own peer ID with the password and sending it together with one's own credentials to the other peer. If the peer ID in the decrypted string and the credentials are the same, the credentials are signed and the signature is returned to the owner which adds the friend (and its signature) to the local friends list. Signatures from friends are used to establish social trust which is explained in Section 3.5.

3.3.3. Update Process

All credentials have a validity period, e.g. 2 years. When the middle of this period is reached, new credentials are generated which lead to a new identity[9]. These new credentials are also signed by the old private key and this signature along with the old ID are included in the history part of the credentials in order to show that both credentials represent the same identity.
This means that friends have a time frame of half the validity period to renew their friendship and sign the new credentials. During that time, the identification can still be done with the old credentials. The update process should be automatic when coming into range not requiring secure pairing unless the particular friendship was intended not to be renewed.

3.3.3.1. Services

Additionally, together with the identity, users may advertise their services in the service mask of the discovery messages. Although no special services are currently available, this service mask could be used in the future to inform other users about special services. A node could for example advertise its Internet connection or a central authority inform others about its registration service. The variety of all these services will be dependent on the users' needs. We currently use the service flag in order to inform other users whether the owner's credentials are trusted, i.e. signed by an authority, or not. Depending on that information, a credential update may or may not be required.

[8]Message that is broadcasted every 2 seconds in order to inform other peers about its presence and services.
[9]Since the identity is deduced from hash of the public key.

Channel Credentials
1. ID (Hash of Public Key)
2. Name
3. Algorithm Identifier
4. Public Key
5. Signature (Over Fields 1-4)

Table 3.3.: Fields of Channel Credentials.

3.4. Authorization

As mentioned in Section 2.1, podcasts are always published into so-called channels. By extending the podcasting concept for peer-to-peer delivery of user generated content, the need for new channel definitions which are more suitable for the new publication scenarios becomes evident. In this section, we define three new channel classes, namely *open*, *restricted* and *closed*, which define different publication and viewing rights for users. In the authorization process, it is defined whether or not a user is allowed to publish into a channel. Each published content is assigned to a channel and an author by including the corresponding channel ID[10] when signing the content. Before certain content is downloaded in a specific channel, the channel properties as well as the channel class and the authorization lists have to be checked to verify whether the transaction is legal.

3.4.1. Channel Properties

Every channel requires credentials which are based on a generated public/private key pair. Similar to the generation of identities in Subsection 3.3.1, the channel is identified by the hash of its public key. The channel credentials and meta data comprise all information about a channel and are both presented in this subsection. Information about access and publication rights of users are all defined in the different authorization lists which are presented in Subsection 3.4.3.

3.4.1.1. Channel Credentials

The channel credentials are listed in Table 3.3. The user holding the corresponding private key is automatically the owner of the channel and allowed to change the channel's meta data as well as all the authorization lists. When owners want to give up their administrative responsibilities in the channel, they could pass the ownership over to another user by sending him/her the channel's private key and deleting it locally.

[10]The channel ID is part of the channel credentials explained in Section 3.4.1 is needed to assure no user can copy the content into another channel where it may not be appropriate thus resulting in a bad reputation for the author.

3.4.1.2. Channel Meta Data

The channel meta data contains additional information about the channel such as the channel policy, a description field, a flag indicating the *channel class*[11], and an optional field disclosing the owner's identity. The meta data is signed with the private key of the channel credentials so that only the channel owner can perform changes to it.

Owner field: The owner field contains the owner's user ID and a signature over the channel ID. The signature assures that the channel is created by the specified user and not by other users trying to profit from another user's reputation.
The field is optional in order to retain the privacy of every channel owner. Although an owner can regulate the content inside a channel with the help of authorization lists[12], he/she may not be able to examine every content published in the channel because of the nature of the distributed environment. Particularly, in *open* channels user may publish content to which the owner does not want to be related to.
On the other hand, disclosing the channel ownership may increase the trustworthiness, e.g. in *restricted* channels because the authenticity can be checked by others.

Channel Policy: If a channel owner wants to specify certain general rules which apply to all users in the channel, he/she can do so by specifying channel policies. Possible policy flags may require the holding of specific signatures (e.g. general CA certification, CA certification with verified age, etc.) or the definition of maximum publication rates[13] per day which could be enforced by the application of the sender itself. However, in the current implementation, no channel policies are defined yet.

Description: All published content in the channel should be related to the description. The description comprises human readable text describing the channel topic. By considering channel descriptions, users can decide whether or not they are interested in the content the channel offers. Optionally, besides the channel title, the description may already be exchanged in the discovery channel serving as a hint for subscription. Instead of human readable text, keywords could be extracted from the description. Such *tags* would be advantageous in comparing and finding similar channels.

3.4.2. Channel Class

We define three different classes of channels depending on the purpose and sensitivity of the published content. They are the *open*, the *restricted* and the *closed* channel. The definitions can be found below.

[11]Comprises *open*, *restricted* and *closed* channels and is explained later.
[12]See Subsection 3.4.3.
[13]In contrast to the spam control mechanism described in Subsection 3.7 which limits the download of content.

Open: In the *open* channel class every user is allowed to read and publish content. This is the most challenging channel class since everybody, even strangers, can publish and therefore needs strict regulations, such as distributed blacklists, ratings and spam control mechanisms. *Open* channels could be used for all kinds of things including discussion boards or interest groups presenting self-made texts, pictures or even videos.

Restricted: In *restricted* channels every user can read but only the members[14] can publish content. No spam prevention is needed because only a limited legitimated group does publish. *Restricted* channels may be used for traditional podcasts from Internet showing publicly available information such as newsfeeds from newspapers or television stations as well as a means to distribute the homework by a teacher in school. These channels are used in case the authenticity of the source is important but the information itself is not secret.

Closed: Private content can be published in *closed* channels. All content in this channel is fully encrypted and only open to members which are listed in the member list[14]. Therefore, closed channels may be used between friends, within a family or in a working environment where informations should not be publicly available. For privacy reasons, contents (including channel meta data and member list) are only exchanged after a successful authentication in order to prove membership. Alternatively some channels could only be advertised in a special environment (see Section 3.5 for more information). The user rights (read/write) inside a channel are regulated in the member list and may still vary between the members. Only the owner or moderators can include members on the member list.
Due to performance issues, symmetric cryptography is applied for encryption. Two methods are theoretically possible: either the channel owner appends a channel key for the encryption of all content to the channel meta data or every author generates its own symmetric key and appends it to the episode meta data. In the current implementation, we use the second method. Independent of the method used, every sender always encrypts the symmetric key with the public key of the communicating member just before sending the actual content. By doing this, the overhead of transferring unneeded keys can be reduced.

3.4.3. Authorization Lists

Authorization lists are tools to specify authorization inside a channel, such as writing or viewing rights for example. As presented in the last section, we defined three types of channels which may have different user roles. The user that created the channel is its *owner* and stores all channel credentials. He/she is able to include *moderators*, which are additional administrators of the channel, on the moderator list. Both moderators and owner have the right put misbehaving users on a channel blacklist and thus ban them from publishing in the channel or, if the channel is restricted or closed, add new *member* on the channel's member list.

[14] See next Subsection 3.4.3.

Both *global blacklist* and *channel blacklist* are part of the *combined blacklist* explained later in Subsection 3.6.3.

Global Blacklist: The global blacklist is only added for completeness. It is made by a central authority and used in a way similar to blacklist in email traffic. Whenever connecting to another peer, it is always checked and exchanged whether the other peer provides a newer version of the list. This blacklist is used globally, meaning independently for all channels. Users which misbehave in many channels by publishing illegal content will be included on this list. Since the list is global it should not become uncontrollably large. For this reason only CA certified identities should be added to this list.

Channel Blacklist: The channel blacklist is only used in open channels where no restrictions on authors exist, in order to prevent malicious users from publishing. The channel owner and the moderators administer this list by adding or removing malicious users which continuously violate the channel rules e.g. by publishing explicit content in a channel for children. Every user inside the channel will automatically receive the updates of this list together with other channel meta data changes and new content whenever he/she downloads from a person having the updated version.

Moderator List: This list contains the moderators of a channel. As explained above, moderators are additional administrators of the channel and can perform different capabilities depending on the channel class. These are:

- In open channels, moderators may update the channel blacklist with malicious user IDs.
- In restricted and closed channels, they are allowed to manage new members.

Only the owner of a channel is able to add new moderators to the moderator list, and for integrity reasons, the list is signed by the channel owner.

Member List: Member lists can only be found in restricted or closed channels because its purpose is to allow certain users to read and/or write content in the channel. Only the channel owner and moderators are allowed to include new members and for integrity reasons, every new entry is signed by its creator. In order to revoke the read and the write rights of a member, his/her list entry has to persist, specifying that he/she has no rights. The member list entries are specified in Table 3.4.
The users on the member, moderator or channel blacklist will eventually change their ID whenever the public key of the user expires. Every moderator/owner who receives an updated certificate for an existing user has to update all the existing entries for that user. All the other users will have to wait for a signed list update from the moderator/owner. However, as specified in Section 3.3.1, users are still able to identify the user with the updated identity by his/her old ID in the history part of the credentials.

Member List
1. User ID
2. Timestamp from Entry Creation
3. Member Rights
4. ID of Moderator/Owner who added the Entry
5. Signature by Moderator/Owner of Fields 1-4

Table 3.4.: All Entries of a Member List.

3.4.4. PodNet Content

Every content, also called *episode* in PodNet, comprises all fields in Table 3.5 consisting of the episode meta data and actual data. All the episode meta data will be downloaded in advance to check for blacklist nominations[15] and ratings before downloading the data. The definition of the meta data was inspired by the MetaInfo file in BitTorrent [4]. The publication time is included for future use and would allow for the downloading and searching of new or recent content. The hash list serves a similar purpose as in BitTorrent allowing to check the authenticity and integrity of the individual data chunks. Authenticity of the content is checked by verifying the author's signature with his/her credentials included in the episode meta data.

Content	
Meta Data	1. ID (Hash over Fields 2-6) 2. Content Name (Human readable and descriptive name) 3. Channel ID (of corresponding channel) 4. Publication Time 5. Hash List of Data Chunks 6. Author's Credentials 7. Properties (File Type, Size, Encryption Flag, ...) 8. Author's Signature (Over ID) 9. Optional: Encryption Key (in closed channels)
Data	Enclosure

Table 3.5.: All Data Fields assigned to an Episode.

If we would rate the suppliers of content instead the authors (explained in Section 3.6), the signature of the content would not be important because an authentication with the supplier is always performed before any data download[16]. The reason for that is the fact that in PodNet everybody downloads content and provides it to others automatically even though the content may not be reviewed yet. Therefore, one cannot hold the supplier accountable for it but only its author.
Whenever a content needs to be deleted, the content's meta data have to persist and be marked

[15]Nomination on either the global, channel, personal or local blacklist, i.e. on the combined blacklist explained later.
[16]See *Challenging Process* in Subsection 3.3.1.

as deleted. If everything would be deleted, the content would automatically reappear whenever a supplier comes into contact.

3.5. Trust Metrics

Trust defines a belief in the honesty and trustfulness of a user's future actions. In [9] and [16], trust is assessed personally by comparing the similarity of personal experiences with prior recommendations[17] received from others. The trust value is dependent on the number and accuracy of ratings. Such a mechanisms would therefore firstly require the users to vote on the same content and secondly would decrease the trust level in users which perform ratings on many authors because the probability of being different is larger.
Since we do not want to expect regular ratings from the users in order for the system to work, we base the trust value of every user on completely different information such as the social relations (friendships), environments and communities. Although the used information is different, the trust values still remain individual and may change from person to person.
In this subsection the difference between community and environment are highlighted. Then the roles a user can play are define in the social model. Furthermore, it is explained how friend circles are built and how communities are detected.

3.5.1. Environment vs. Community

Although an environment is always associated with geographic locations and a community is linked to people, both are closely related to each other. The relation is based on the assumption that at a different location, one is surrounded by different people.
Similar to [28], the environment can be classified into three categories: *home*, *work* and *other*. In each of them, different security mechanisms may be applied. Automatic detection of the environment is hard since in the absence of any specialized location-aware hardware, a device usually does not know its exact location. One could possibly try to analyze the neighbor devices present at different times of the day. However, the devices would still not know their exact location but only their neighboring devices. The environment could be completely different as for example at work or taking a break with some colleagues at a bar. Therefore, we abandon environment classification and focus on community detection in the following.
A community can be described as a well linked clustering of entities [29]. An entity is not bound to one specific community and may therefore belong to more than one. Additionally, an even more closely linked part of a community may build a sub-community. By analyzing contact duration and/or contact frequency of the surrounding peers and sharing this information with those, one can try to guess the communities to which one belongs.

[17]Recommendations about other user's behavior. It will be defined later in Section 3.6.

3.5.2. Social Model

Before going into details on community detection, the social model in PodNet is explained, highlighting the different user roles in the system. Some roles may be combined since they are not necessarily mutually exclusive.

Neighbors: Any peer that is in the user's proximity at a given time is his/her neighbor. As there is no special requirement on neighbors, they can be combined with any other role. Encountered neighbors are saved for a certain time, depending on the regularity and the duration they are encountered as well as an aging factor. See Subsection 3.5.4 for more information.

Authors: An author is the producer of content. Identifying authors is important in order to hold users accountable for what they publish even though they are not necessarily neighbors. Authentication can be performed by checking the author's signature included in the episode meta data[18] and accountability is fulfilled by applying the rating mechanism which is described in Section 3.6.

Suppliers: Suppliers are users who provide content to other users and hence are neighbors. However, they are not necessarily the authors of the content and therefore, cannot be held accountable for it.
Optionally, in order to save resources, a supplier could decide to spread content only for a certain time or with a certain probability. Besides that, the usage of incentives to reward suppliers could motivate people to provide content and ensures fairness in the distribution process. However, such optional features are out of the scope of this work.

Friends: Friendship is a relation with a trusted peer which is established through secure pairing as explained in *Secure Pairing* in Subsection 3.3.2. It can only be performed consciously when both peers meet each other (neighbors) and agreed on a common password for the secure pairing. After having established friendship, these peers will trust each other to a much higher degree. In particular, this influences the social trust value which is important when exchanging ratings as explained in Subsection 3.6.2. Besides that, friendship relations help building up the friend circle as explained in the next paragraph. In contrast to [6] and PGP, signatures from friends are not required for usage but are an add-on in order to increase the social trust.

Friend Circle Members: When two peers meet, they may first exchange their friends list and then check whether they have common friends. As shown in [30], friends up to six hops can be trusted to a high degree. Based on the common friends information, friends circles can be built as explained in Subsection 3.5.3. Members of a friend circle are assumed to be more trustworthy than regular users but less trustworthy than direct friends.

[18]See Subsection 3.4.4 for more information.

Familiar Strangers: Familiar Strangers [31] are people we do not know (strangers) but meet regularly (familiars). They can be more trusted than completely unknown users due to two reasons. Firstly, these peers do not frequently change their identity in order to abandon bad reputation. Secondly, incomplete files can be easily completed at another day. Although an attacker could still initially behave nicely and start an attack some weeks later after having an increased trust level (and also an increased potential effect), the weight given to these peers is still low and generating such identities would require a much higher effort for the attacker. In order to prevent attackers from using several identities at the same time, some techniques such as in [20] could be used[19]. For the sake of simplicity, we make no distinction between the terms *Familiar Strangers* and *Familiars* and use them interchangeably. If *familiar* is used as a predicate e.g. for node, user or peer, it implies that the corresponding node, user or peer is a *Familiar*.

Community Members: Users that have a high percentage of common *Familiar Strangers* are assumed to be well linked and are thus in the same community. Communities can be formed at locations where most people see each other over a significant period of time, e.g. in work environments or at home.
Community members are more trusted than single *Familiar Strangers* since they share a significant part of daily and social activities. However, they are still less trusted than *Friends* or *Friend Circle Members* because an attacker can always obtain community membership without much effort assuming he/she is a familiar already as shown in Subsection 6.4.1.

Strangers: Unknown users are classified as *Strangers*. Initially every user is a stranger because no or only little information about the user is available. Therefore, *Strangers* are the least trusted users in the network because they have not yet proved to be honest. Strangers are usually new users of the network, which have only rarely interacted with others. However, attackers which do often change their identities in order to obfuscate their intentions may also be classified as strangers. By meeting other people regularly, a stranger can become a familiar peer, community member and by secure pairing build friendships and friend circles thus increasing his/her social trust value for specific peers.

3.5.3. Friend Circle Buildup

In order to build up a friend circle, a node n_i requires to be friends[20] with at least one person who is saved in $n_i's$ friends list FR_i. As shown in Algorithm 2 a user's friend circle contains at least all the 2-hop friends, i.e. the friends of friends. So any two peers that have a common friend will always add each other to their friend circle. Additionally, a peer is added as well, if at least a certain adding percentage p_{add} of their friends are already in one's friend circle.
Additionally, the peer's friends are merged into one's friend circle if they coincide at least by the merging percentage p_{merge} because we assume that those peers may have similar trustability

[19]See *Sybil Attacks* in Section 2.5.
[20]The process of becoming friends is described in Section 3.3.2 under *Secure Pairing*.

Algorithm 2 Friend Circle Buildup

```
$p_{add}$: Adding threshold
$p_{merge}$: Merging threshold
$n_i$: A node
$FR_i$: Set containing all friends of node i
$FC_i$: Set containing all friend circle members of node i
$n_0$: Local node
$FC_0 = \emptyset$
for all nodes $n_i$ in proximity do
    acquire $FR_i$ from $n_i$
    if $n_i \in FR_0$ then
        merge $FR_i$ and $FC_0$
    else
        if $FR_i \cap FR_0 \neq \emptyset$ then
            add $n_i$ to $FC_0$
        else if $\mid FR_i \cap FC_0 \mid / \mid FR_i \mid > p_{add}$ then
            add $n_i$ to $FC_0$
        end if
        if $\mid FR_i \cap FC_0 \mid / \mid FR_i \mid > p_{merge}$ then
            merge $FR_i$ and $FC_0$
        end if
    end if
end for
```

as those already in the circle. Note that unlike friendship established through secure pairing, the friend circle adding and merging process is not necessarily symmetric as explained in the example below.

The probability of merging or including a user into the friend circle decreases with increasing hop distance and thus limits the size of the friend circle and the order of transitivity.

An example scenario is depicted in Figure 3.2. For this example we chose a p_{add} of 60% and a p_{merge} of 90%. The nodes *Red*, *Green* and *Blue* are shown together with their friends (the smaller nodes in the circle around the them). Additionally, the *Red's* friend circle is shown which contains some of *Green's* and *Blue's* friends (orange and oragnge shadowed nodes). In case *Red* would meet *Green*, he is added to the friend circle since more than 60% of his friends are in *Red's* friend circle.

In case *Blue* is met, *Red* would not only add him to the friend circle, but also merge *Blue's* friends, since 90% of them are member of the friend circle already. On the other hand, *Red's* friends might not be in the *Blue's* friend circle which would give *Blue* no reason to add *Red* to his friend circle.

3.5.4. Community Detection

As mentioned in Subsection 2.6, not much work has been published for distributed community detection in delay-tolerant networks. In [24] the three algorithms *Simple*, *k-Clique* and *Modular-*

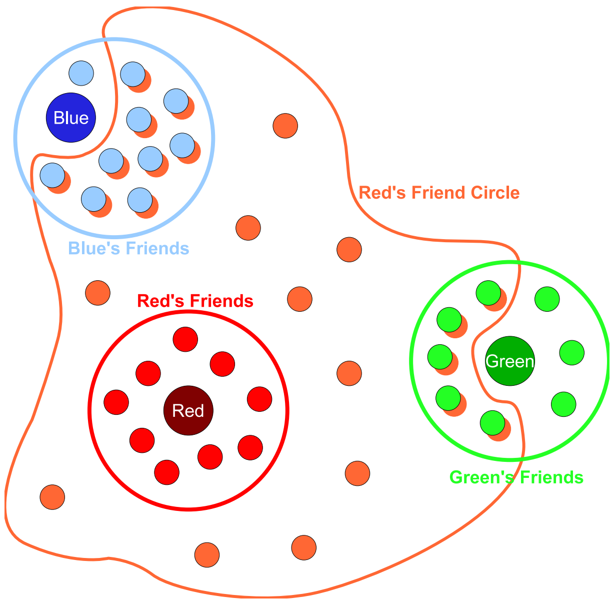


Figure 3.2.: A Friend Circle Scenario.

ity are proposed.
The *Simple* algorithm needs the least overhead in computation, storage and sending but suffers the disadvantage that an accelerated community construction by merging communities is very inaccurate. Every user only exchanges his/her familiar and community set.
Both *k-Clique* and *Modularity* algorithm need high sending and storage overhead since additionally all familiar sets of all community members are also exchanged. On the other side, both algorithms can perform an accelerated community detection by community merging without losing the construction precision since all the community member information is available as if the respective peers would be present.
Compared to *Simple* and *k-Clique*, the *Modularity* algorithm has the advantage that (besides the total contact duration threshold for familiar nodes) no additional design threshold is needed, making the algorithm more flexible for changing environments. However, it needs much more processing overhead because the modularity has to be re-calculated for every community member whenever a new community member is added.

It is evident that all three algorithms have advantages and disadvantages. We decided to base our community detection algorithm on *Simple* because of the fewer computational and sending overhead which is crucial on mobile devices[21]. In [24] they also suggest the *Simple* algorithm for mobile devices with constraints on complexity because it showed quite acceptable results.
However, this algorithm is not perfect at all and more research is needed in this area, especially in considering the aging aspect as already mentioned in Section 2.6. For this reason we have

[21]Compare to *Performance Requirements* in Subsection 3.1.

slightly modified the algorithm in several ways.

3.5.4.1. Modified Community Building Algorithm

The main change to the *Simple* consists of substituting the community adding criterion in Algorithm 1 from

$$| F_i \cap C_0 | \;/\; | F_i | > \lambda \tag{3.1}$$

to

$$| F_i \cap F_0 | \;/\; | F_i | > \lambda \quad \wedge \quad | F_i | \geq k \tag{3.2}$$

where λ is the adding ratio, F_i is the familiar set of the other node and F_0 and C_0 are the own familiar and community set respectively. Additionally, similar to the *k-Clique* algorithm, we require a minimum of k members to be in the familiar set before the community algorithm considers adding the peer.
By this modification the quality of the detected communities are improved significantly. The evaluation of the modified algorithm can be found in Section 6.4.1.

3.5.4.2. Aging Mechanism

The aging mechanism we apply consists of two parts, a *Familiar Set Aging* and a *Community Set Aging*, which should regulate the size in the familiar and community set respectively. Whenever two devices see each other, their contact duration value is increased. The *Familiar Set Aging* mechanism decreases that contact duration value periodically.

Parameter	**Description**
f_size_{min}	Min. familiar set size
f_size_{max}	Max. familiar set size
f_age_{min}	Min. familiar set aging speed
f_age_{max}	Max. familiar set aging speed
f_age_{step}	Familiar set aging speed step
f_th_{min}	Min. familiar set adding threshold
f_th_{max}	Max. familiar set adding threshold
f_th_{step}	Familiar set adding threshold step
f_age	Familiar set aging speed
f_th_{add}	Familiar set adding threshold
f_th_{rem}	Familiar set removing threshold
c_size_{min}	Min. community set size
c_size_{max}	Max. community set size
c_th_{max}	Max. community set removing threshold
c_th_{step}	Community set removing threshold step
c_th_{rem}	Community set removing threshold
c_age	Community set aging speed
x_{alien}	Alienation factor

Table 3.6.: Description of aging algorithm parameters.

In order to have more dynamic parameters, we decided to make the familiar set aging speed f_age as well as the adding threshold f_th_{add} (defines when a peer is added to the familiar set), dependent on the familiar set size. Similar to [29], the aging should depend on the time passed since the peer was last seen. Therefore, the aging is multiplied by a factor that increases exponentially with time in order to neglect short time variations. We call this the alienation factor x_{alien}. The description of all the required parameters[22] are summarized in Table 3.6.

Algorithm 3 Familiar Set Aging

```
$f\_age = f\_age_{min}$
$f\_th_{add} = f\_th_{min}$
$f\_th_{rem} = f\_th_{min} - f\_th_{step}$
loop
  for all nodes $n_i$ in the familiar set $F_0$ do
    reduce $n_i's$ contact time $t_i$ by $f\_age \cdot x_{alien}$
    if $t_i < f\_th_{rem}$ then
      remove $n_i$ from $F_0$
    end if
  end for
  if $| F_0 | < f\_size_{min}$ then
    reduce $f\_th_{add}$ and $f\_th_{rem}$ by $f\_th_{step}$   (if $f\_th_{add} > f\_th_{min}$)
    reduce $f\_age$ by $f\_age_{step}$   (if $f\_age > f\_age_{min}$)
  else if $| F_0 | > f\_size_{max}$ then
    increase $f\_th_{add}$ and $f\_th_{rem}$ by $f\_th_{step}$   (if $f\_th_{add} < f\_th_{max}$)
    increase $f\_age$ by $f\_age_{step}$   (if $f\_age < f\_age_{max}$)
  else
    if $| F_0 |$ increased since last aging cycle then
      increase $f\_th_{add}$ and $f\_th_{rem}$ by $f\_th_{step}$   (if $f\_th_{add} < f\_th_{max}$)
      increase $f\_age$ by $f\_age_{step}$   (if $f\_age < f\_age_{max}$)
    else if $| F_0 |$ decreased since last aging cycle then
      reduce $f\_th_{add}$ and $f\_th_{rem}$ by $f\_th_{step}$   (if $f\_th_{add} > f\_th_{min}$)
      reduce $f\_age$ by $f\_age_{step}$   (if $f\_age > f\_age_{min}$)
    end if
  end if
  wait for next aging cycle
end loop
```

The idea behind having an aging factor depending on the familiar set is to give less social users the ability to build small communities, as well as to limit the community size for very social users. In order for the aging to work for a broad range of social behaviors, the range of desired familiar set sizes is specified between f_size_{min} and f_size_{max}. As long as the familiar set size is below f_size_{min} the aging speed f_age decreases each aging cycle towards a minimum f_age_{min}. While the size of the familiar set is in the desired range, f_age is augmented or reduced depending on whether the familiar set size has increased or decreased since the last aging cycle. As soon as the size surpasses f_size_{max} the aging speed f_age increases dramatically

[22]An example of specific values for the parameters can be found in Table E.20.

towards a maximum f_age_{max}. The familiar set adding threshold behaves exactly the same as the aging speed. The *Familiar Set Aging* mechanism is described in Algorithm 3.

Algorithm 4 Community Set Aging

```
$c\_th_{rem} = c\_th_{max}$
**loop**
  **for all** nodes $n_i$ in the community set $C_0$ **do**
    increase $n_i's$ community member time $t\_cm_i$ by $c\_age$
    **if** $t\_cm_i > c\_th_{rem}$ **then**
      remove $n_i$ from $C_0$
    **end if**
  **end for**
  **if** $\mid C_0 \mid < c\_size_{min}$ **then**
    increase $c\_th_{rem}$ by $c\_th_{step}$    (if $c\_th_{rem} < c\_th_{max}$)
  **else if** $\mid C_0 \mid > c\_size_{max}$ **then**
    decrease $c\_th_{rem}$ by $c\_th_{step}$
  **else**
    **if** $\mid C_0 \mid$ increased since last aging cycle **then**
      decrease $c\_th_{rem}$ by $c\_th_{step}$
    **else if** $\mid C_0 \mid$ decreased since last aging cycle **then**
      increase $c\_th_{rem}$ by $c\_th_{step}$    (if $c\_th_{rem} < c\_th_{max}$)
    **end if**
  **end if**
  wait for next aging cycle
**end loop**
```

The community set is aged as well removing community members after a certain time of complete inactivity. This is done by increasing the timeout of each community member during every aging cycle by c_age. In case the timeout reaches the removal threshold c_th_{rem} the corresponding user is removed from the community. Every time a community member is met and fulfills the community adding or merging criterion, the timeout of the community member is reset to zero. As can be seen in Algorithm 4, the removal threshold c_th_{rem} depends on the community set size similar to the familiar set adding threshold f_th_{add} is depending on the familiar set size as described above. This *Community Set Aging* algorithm requires the range of a desired community set size to be defined in between c_th_{min} and c_th_{max}. Its behavior below, in and above this range is comparable to the *Familiar Set Aging* mechanism.

3.5.4.3. Optional Modification

Additionally, one could think about not using the accelerated community construction by disabling the merging of communities. The reason for this is that it is very inaccurate and community detection should be done individually for every node. In [32], it is observed that most people are members in several communities which may differ in size and people. Very active people which are in several different communities, e.g. home, work, sports club, politic party, etc., would hardly ever merge a community because the percentage of common people would be

small.
A positive side effect of avoiding the merging of communities is that the exchange of the community set would become obsolete which would decrease the sending overhead and computational complexity.

3.5.5. Trust Weights

Ideally, if there were no malicious users and if no information would be available, all suggestions, even from unknown users would be helpful. However, if information from trusted users is available, the influence of unknown users should be limited so that they can hardly outstrip the more trusted users. There is always a tradeoff between relying on opinions from unknown persons when no information is available and neglecting this information completely.
In Table 3.7, we present the weight assignment we chose in our design. However, depending on the preferences (security versus information) the weightings could easily be changed in the future.

Group	**Trust Weight**
Stranger	1
Familiar Stranger	5
Community Member	10
Friend Circle Member	25
Friends	50
Oneself	100

Table 3.7.: Trust Weights.

In [23], the cost of trusting unknown users when allowing users to freely change identities, is formally analyzed using game-theoretical models. Even though they concluded, that in such a case, all unknown identities should be regarded as malicious, we will not neglect these users but give them much lower weight. Since ratings are only exchanged between directly connected peers, an attacker would have to connect to every victim that he/she wants to fool with much more fake identities in order to exchange fake ratings and thus locally manipulate the victim's perception of another user's reputation.

3.6. Reputation System

The reputation of a user regards his/her behavior norms and is based on past experiences. It combines *personal opinion* and received *recommendations* which are both obtained by rating the author.
Personal opinion comprises the user's own view from his/her own past experiences about the content from an author. The reputation of an author is built by calculating the weighted sum of one's own opinion and received recommendations from others. Note, that in order to avoid loops and the resulting rating amplification, we only exchange information from direct experiences

when meeting another peer, i.e. own download experiences. The weights of the recommendations are independent of the user's rating and depend completely on their trust value which is defined in Section 3.5. The reputation of an author should help a user to decide whether or not to download content from that author.

Every user can assess an author's reputation by a two-level rating which measures the legitimacy and satisfaction of published content per author and channel (explained in Subsection 3.6.1). The author can thus have different reputations values in different channels and the ratings evolve over time as old ratings have a decreased influence on the reputation.
In [9], it is stated that reputation should comprise the following four properties: subjectiveness, context-dependence, dynamics and time. In the following, we will analyze our proposed rating against these four properties.

Subjectiveness: The rating is subjective because the same content can be rated differently by different users. However, it is not arbitrary and therefore we use the two level rating which on the first level measures the legitimacy (objective rating, should be similar for all users) and a subjective rating which may be different depending on the users and his/her taste.

Context-Dependence: The context-dependence of a reputation is important since e.g. a good engineer may not be automatically a good cook. We consider context-dependence because all reputation and ratings of an author is always relative to a specific channel. Instead of content rating, we apply author rating because we assume that the behavior of an author inside the same channel will not vary significantly.

Dynamics: The reputation should be dynamic in order to allow bad authors to improve their reputation and good authors should get a lower reputation when publishing bad content. This will be ensured by using a fading factor smaller than 1 in the personal reputation calculation shown in Equation 3.3.

Time: The time requirement regulates the aging of all ratings so that old ratings are less weighted than new ratings. We currently don't consider explicit aging of rating weights as in [9]. The fading factor naturally ages the personal ratings implicitly and the validity of blacklist entries is regulated by timeout values defined in Subsection 3.6.4. We decided not to age the rating values because the rating should be valid as long as the content is present.

3.6.1. Personal Opinion

In order to express an opinion about content, a user has to rate the content. The corresponding author is then added to the personal list which should reflect this opinion. More importance should be given to ratings of recently obtained content in order to calculate an up-to-date opinion of the author.

3.6.1.1. Rating

Every user may optionally rate content to influence the reputation value of its author locally and as a consequence improve the quality of the content received in the future. As indicated in Section 2.3 many different rating system exist in literature. In most systems, the user is asked for an opinion (legitimacy, quality, taste) about a content or service and the system deduces an appropriate action from it. In this work, we will use a two level rating which will evaluate legitimacy and quality/taste of the content at the same time.

The first level comprises *objective* rating, i.e. whether the content is legitimate. The second level rating is *subjective* and is used to rate either the quality of the content or reflects the user's taste. It is difficult to differentiate between quality and taste because a user's taste may influence his/her perception and priority of the quality. Both rating levels are mutually exclusive meaning that if a content is rated as not legitimate, no second level rating is needed and vice versa.

As specified above, the author rating is context-dependent relative to the channel the content is published in. All the personal ratings (subjective or objective) from direct experiences are stored in *personal lists* and will be exchanged among all users subscribed to the same channel prior to every data exchange inside the channel. The users that receive that information will store it in their *local lists* and process it based on the trust value of the recommender as explained later in Subsection 3.6.2.

Objective Rating: An objective rating states whether content is legitimate or not. If the content is not legitimate, it might be one of the following:

- illegal/objectionable
- fake
- spam
- misplaced

Non-legitimate content such as illegal/objectionable, fake or spam should be blocked because it is clearly unwanted content. It is arguable whether misplaced content should be treated as harsh. More appropriate might be to delete it locally and rate it as (subjectively) bad.

Subjective Rating: Content can be rated as either good or bad depending on the quality or the taste. The number of consequent good/bad content in the personal rating list will influence the spam control mechanism (see Section 3.7). When several of the received contents from the same author are rated as bad, the author can be locally blocked without notifying other users by setting a blocking flag in the personal rating list. In this case, only a subjective bad rating would be exchanged with others.

The subjective rating is translated from ‘good’ and ‘bad’ into ‘+1’ and ‘-1’. We use binary values for three reasons: Firstly, we want all the values to have the same influence on the rating which would not be the case with multi-level ratings. Secondly, we assume that a user will only rate a content when he/she either liked or disliked it. Contents that are just acceptable may not be rated at all. Thirdly, the rating should influence the binary decision whether to download content or not. Since the decision should be taken automatically, a binary rating may simplify that mechanism. This rating value r is then added to the existing rating r_{old} the following way:

$$r_{new} = r_{old} \cdot a + r \cdot (1 - a) \tag{3.3}$$

The parameter ‘a’ denotes a fading factor. It may be in the range of [0, 1[. The closer ‘a’ is to 1, the more important are the previous ratings, i.e. the slower the fading. In order to illustrate a concrete example, for a fading value of 0.9 the formula would be:

$$r_{new} = r_{old} \cdot 0.9 + r \cdot 0.1 \tag{3.4}$$

This personal rating is performed exactly the same as in [9] without the time dependency of the fading factor. The resulting rating may be in the continuous range of]-1, +1[. The higher the absolute value of the rating, the higher the confidence in the rating. Similar to [9] one could even derive a multi-level rating by defining e.g. values larger than 0.2 as trustworthy, values larger than 0.8 as very trustworthy and similar for negative ratings values smaller than -0.2 untrustworthy and smaller than -0.8 very untrustworthy. All values are initialized with zero which means ignorance.

The subjective rating influences the spam control mechanism as described in Section 3.7. If the subjective rating reaches a certain (negative) threshold, the corresponding author is blocked for a period of time depending on the *Blocking Time* (explained in Subsection 3.6.4).

Blocking Content: Although blocked content is not automatically deleted[23], the content itself and all following contents from the same author are prevented from being exchanged at all. Blocking content due to personal taste as with the subjective rating (explained above) should not affect the overall spreading performance of the content. Indeed, results in [33] and our simulations with non-cooperating nodes in Section 4.4 indicate that limiting the number of suppliers only slightly reduces the overall spreading performance.

On the contrary, malicious content which is blocked due to objective rating should be prevented from spreading. Therefore, this blocking information is exchanged to all users subscribed to the same channel, which, on their part, would start blocking the content as soon as a sufficient

[23]This decision is left to the user.

amount of blacklist suggestions are received[24], i.e. more than the *suggestion threshold*.
There are two distribution processes fighting against each other, namely, the local blacklist and content spreading. By introducing the *suggestion threshold*, the blacklist spreading is put at disadvantage compared to the content spreading because the reception of several blacklists is needed before the measure becomes active. However, the spreading of blacklist information has a clear advantage when several malicious authors generate content because the blacklist information can be summarized in only a few packets and are sent at the beginning of the data transfer, whereas content exchange needs much more time.

Assuming an author to regularly send spam using the same identity over a long period of time, the users would have time to rate content and exchange ratings so that the spreading of future contents could be prevented. Normally, there is a delay between content reception and content rating which leads to a problem if an author just wants to flood the channel with many contents at the same time and then disappears forever. Therefore, we have applied a spam control mechanism which regulates the amount of content a user receives from different authors. The rate limiting is explained in Section 3.7.

3.6.1.2. Personal Lists

Every device stores its personal ratings either in its rating or blacklist depending on whether the rating was subjective or objective.

Personal Blacklist: The personal blacklist contains all the authors that should be blocked as a consequence of an objective rating. The list also includes a counter c_{bl}, stating how many times an author has been blocked, as well as a timeout t_{block}, indicating how long the particular author will be blocked. Each time the counter increases, the blocking time will increase. See Subsection 3.6.4 for more information on the blocking timeouts. This adaptive mechanism tolerates rare misconducts but remembers the past misbehaviors similar to [34]. The blacklisted entries are exchanged with others as long as they are blacklisted locally.

Personal Rating List: The personal rating list contains two values for each author: the rating itself and a spreading timeout value which indicates how long the direct rating is spread. The timeout is necessary for two reasons: firstly, only recent ratings should be spread and secondly, the amount of sent ratings should be limited.
Some informations about the number of received ratings may be useful. However, the calculated rating value indicates whether several rating exist and whether the majority of the recent ratings about the author are positive or not.

[24]Note that only direct information from own experience with downloaded content is exchanged. A node which starts blocking a peer because of received blacklist information will not share this information with others. See Subsection 3.6.2 for more information.

3.6.2. Distributed Recommendations

Having a personal opinion about some content and its author is a good way to increase the quality of the received content. However, ideally, one would like to know something about the quality of a content produced by an not yet known author before actually getting it, i.e. before having an own opinion. This is the reason why recommendations from other users become important. Every user has to distribute its own personal opinions to its neighbors and all received personal opinions (i.e. recommendations) from others are summed up and stored in local lists (described below). We assume the exchanged overhead to be small since all rating and blacklists are always exchanged within a context (i.e. in every channel) and therefore, the recommendations affect all subscribed users.

The reputation of an author is the combination of all recommendations received from users one has interacted with (which are stored in the local lists), and, if available, the own personal opinion (which is stored in the personal lists). In this subsection the distribution process and the concept of local lists is described.

3.6.2.1. Distribution Process

When two users interact, the personal rating and blacklists are exchanged. In order not to receive the same list too many times, the exchange is only performed when the list changes which can be regulated by a timestamp.
Only basic information from the personal lists is exchanged, i.e. for the blacklist the identity of the blocked author, and for the rating list, the author ID together with a discretized rating value[25].

3.6.2.2. Local Lists

The received recommendations are stored in two local lists. The blocked authors are added to the local blacklist and the ratings are integrated into the local rating lists.
We don't need rating tuples as in [16] because every user has discretized his/her own rating values into a four level rating depending on his/her personal rating list. This value is already aggregated over several contents and indicates confidence by its absolute value.

Local Blacklist: The local blacklist contains an entry for every author a blacklist recommendation was received. Every entry itself contains a list of all the user that have recommended to block the author and a timeout value for each recommendation. If the number of recommendations exceeds a certain *suggestions threshold*, the author is blocked until the blocking timeout is reached. See Subsection 3.6.4 for more information on the timeout length. Recommendations from different sources are weighted differently depending on their social trust value. This would

[25] For simplicity reasons we transform the continuous range of the rating to a value in the discrete set $\{-1, -0.5, +0.5, +1\}$ as explained later. The higher the absolute value, the higher the confidence in that value.

mean, assuming a suggestion threshold of 100, either 2 friends, 4 friend circle members, 10 community members, 20 familiars or 100 strangers need to provide a blacklist recommendation in order for an author to be blocked. The complete list of trust weights can be found in Subsection 3.5.5. Optionally, as a potential future improvement, the number of needed recommendations may depend on the personal rating of the author in question. This would mean that if one has a good opinion about an author more suggestions are needed before he/she gets blocked.

Local Rating List: The local rating list is a list comprising all the received ratings. The received ratings comprise values from the personal rating lists of the corresponding sender. The continuous value is discretized to a discrete value in the set of $\{-1, -0.5, +0.5, +1\}$ in order to save resources in sending and storing. The higher the absolute value of the rating, the higher the confidence of the rater in his/her rating. A user cannot gain higher influence by sending its rating values more often since his/her old rating values are replaced by the new ones if they have changed. The influence in the combined reputation is only dependent on the social trust value of the recommending user.

3.6.3. Combined Reputation

In order to get a usable reputation value, the personal and the local lists have to be combined. Every author which is on any of the two blacklists is blocked. An overview over all values that influence the combined blacklist can be found in Figure 3.3.

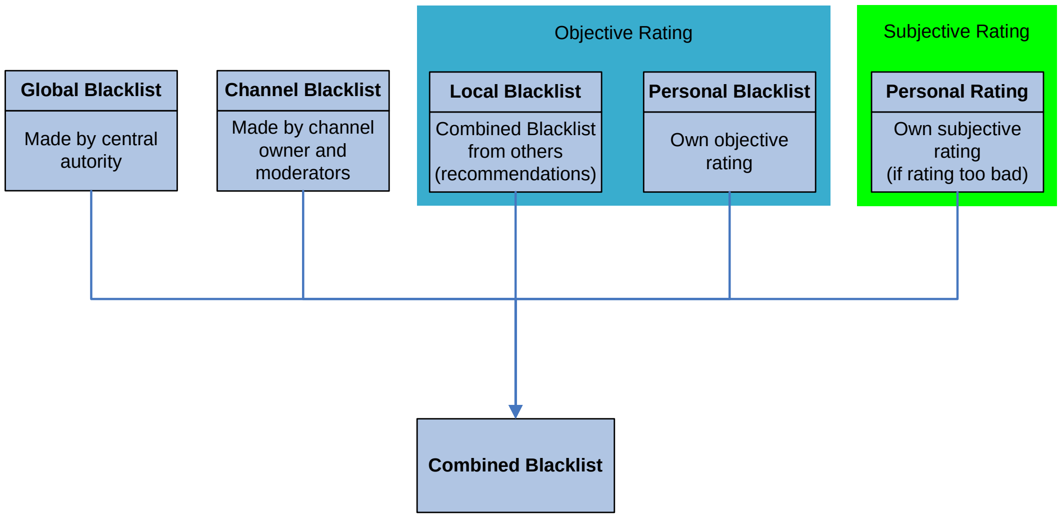


Figure 3.3.: All Influences on the Combined Blacklist: *The combined blacklist is build from the global and channel blacklist (which are both central), from own and received objective ratings as well as from own subjective ratings if the rating is too low. Note that the entries from the global blacklist are used locally and only objective ratings from the personal blacklist are exchanged with others.*

The combined rating is the weighted sum of all the ratings for a particular author, including the ones in the local and the personal rating list. The different entries should be weighted differently according to the trust weight of the rater. Figure 3.4 gives an overview over the combined rating.

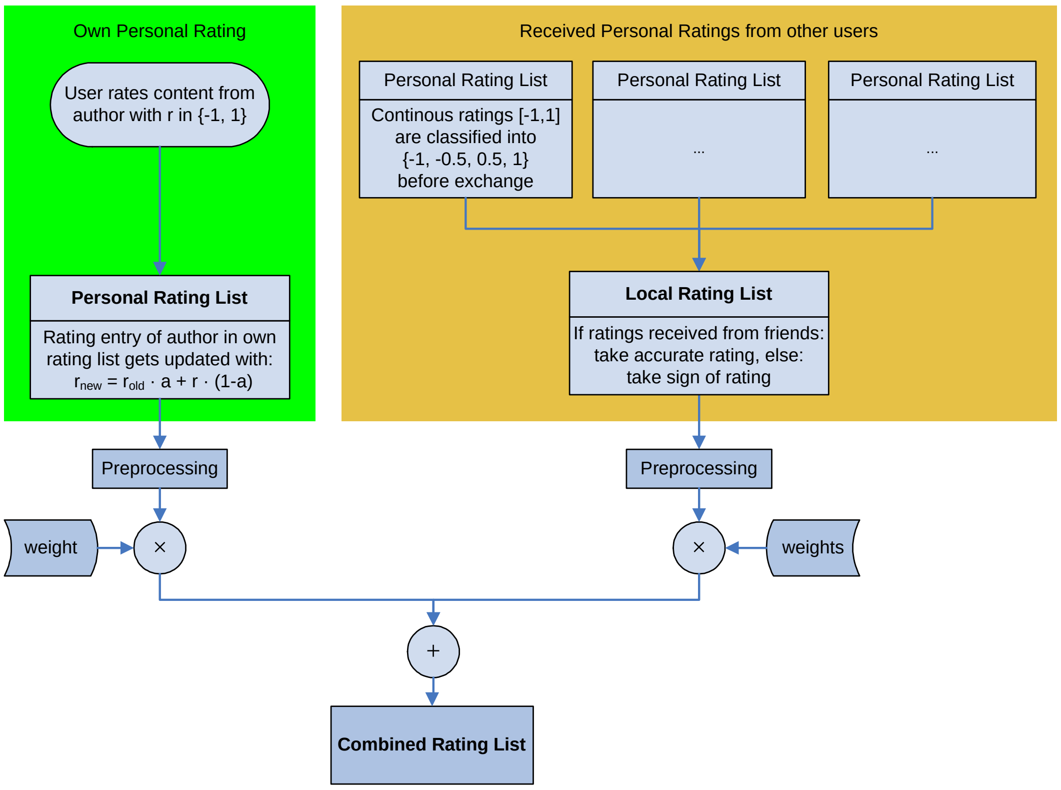

Own Personal Rating
User rates content from author with r in {-1, 1}
Personal Rating List
Rating entry of author in own rating list gets updated with: $r_{new} = r_{old} \cdot a + r \cdot (1-a)$
Received Personal Ratings from other users
Personal Rating List
Continous ratings [-1,1] are classified into {-1, -0.5, 0.5, 1} before exchange
Personal Rating List
...
Personal Rating List
...
Local Rating List
If ratings received from friends: take accurate rating, else: take sign of rating
Preprocessing
Preprocessing
weight
×
×
weights
+
Combined Rating List

Figure 3.4.: All Influences on the Combined Rating List: *The combined rating list is build from the personal and local rating list. The entries are first preprocessed to ensure that the minimum influence of trusted accurate ratings is at least as high as of untrusted ratings. The resulting rating values are then weighted according to their trust values and combined to a weighted sum.*

Before building the actual weighted sum the ratings have to be preprocessed for the following reason: In case several ratings from several sources with different trust values are available the simple weighted sum would produce a reasonable value. But if only a stranger's rating is available and this stranger gives the author the maximum value the weighted sum would produce a high value we cannot really trust. Therefore the influence of such ratings has to be limited. It should not be possible for a user to increase his/her influence with a higher (positive or negative) rating. As a consequence we only consider the sign $(+/-)$ of the ratings received from lesser trusted users. This results in an influence range between -1 and 1.
On the other hand, ratings received by friends are trusted to be accurate thus all four discrete values are used. However, we would like the minimal influence of a friend to be the same as a strangers influence. For this reason we multiply their rating by 2 resulting in the values -2, -1, 1 and 2. Finally, our own rating minimal influence should be adjusted to the strangers influence range as well and is therefore multiplied by 10, producing a value in the continuous range between -10 and 10.

It might be a reasonable idea to share a more fine grained rating, e.g. 10 discrete values instead of 4, and to differentiate more between the trust levels. An example would be to use all the 10 discrete values for friends, 8 for friend circle members, 6 for community members and so on. However, due to simplicity reasons, we currently avoid using such a fine grained rating.
All the trust weights used to combine the ratings are summarized in Subsection 3.5.5. As we mentioned in Section 3.5, we use static weights which depend on the user's social relations and environment. We do not adjust weights based on the accuracy of their recommendations as in [9] and [16] since we may not have enough common ratings (personal and recommendations) to assess a reasonable value.

The advantage of the weighted sum is the fact that it preserves the rating ratios depending on the weights. Because of the additional preprocessing of the ratings, if we see a rating value higher than 1, we know for sure that rating was influenced by at least one friend who is pretty confident in his rating or by oneself. If a rating value is higher than 2, we can directly conclude that the rating has to be influenced by oneself which gives higher confidence in the rating.

In case one has rated an author positively once, the resulting personal rating would be

$$(0 \cdot 0.9 + 1 \cdot 0.1) = 0.1. \tag{3.5}$$

With the described combination process the resulting preprocessed value would be 1 and in order to compensate that rating one would need 100 strangers, 20 familiars, 10 community members, 4 friend circle members and 2-4 friends which have a contradicting opinion. If one has rated more than once, more contradicting ratings are necessary.

The combined rating currently only influences the duration of the *Blocking Time* (see next subsection). However, in future applications, the combined rating may be useful when regulating the amount of content that should be downloaded by the rating quality as explained in Subsection 3.8.2.

3.6.4. Aging and Timeouts

There are four aspects of an authors reputation entry that should age. Firstly, the validity of a list entry should be limited, secondly, the personal opinion about an author should only be spread for some time, thirdly, an author should not be blacklisted indefinitely and fourthly, authors having bad ratings should not be blocked forever.

Author Lifetime: An author entry should not be valid forever. If the entry does not change at all (lacking new personal or reputation ratings), it could be deleted after some time which could depend on the rating the author has and whether his/her content is locally available. If the rating is very good or very bad, the lifetime could be longer, since the high rating value of the author must have required many episodes that have been rated and remembering this information may be more important. In the current design we delete the entry after having deleted all of his/her content and waiting an additional timeout period t_{author} and depending on the author rating r_{author}:

$$t_{author} = 28days \cdot r_{author} \tag{3.6}$$

We chose a minimum of 28 days (four weeks) of remembering because people may not rate that often and we don't want to forget authors too fast.

Spreading Timeout: Whenever a user creates or changes the personal rating of an author, this opinion will be shared with other users but only for some limited time in order to reduce the sending overhead of the channel meta data. Assuming a user having an almost fixed environment of people he/she regularly sees, one can assume that the rating and content is already exchanged and one could save power and time by not sending old ratings again. In our prototype, we chose a spreading timeout t_{spread} of

$$t_{spread} = 14days \tag{3.7}$$

which correspond to two weeks. Even if a user is met regularly the information could still be missed the first week when being sick or absent for a few days. This user could receive it in the second week. Note that the spreading timeout only affects personal ratings since blacklisted authors are exchanged as long as the authors are blacklisted.

Suggestion Timeout: As mentioned in Subsection 3.6.2, information about blacklisted authors is received from other users. All received blacklist recommendations are stored and if the number of entries exceeds a certain suggestion threshold, the author will be blocked. However, the recommendations should only be remembered for a short time because of people who deliberately try to blacklist specific authors by sending blacklist recommendations. The timeout is reset whenever the recommendation is received again from the same author. In the current design, we assume a suggestion timeout t_{sugg} of

$$t_{sugg} = 7days \tag{3.8}$$

which corresponds to one week. We chose it as a tradeoff between wanted information spreading and limiting unwanted wrong recommendations. Assuming a fixed environment, where everybody sees everybody at least once a week, the recommendations would be reset within the timeout period whereas fake recommendations which may not be met regularly may be removed.

Blacklisting Timeout: There are two different blacklists: personal and local blacklists. Authors being on either one of both are blocked for a certain period of time. The first time the author is blacklisted, the blocking period should be shorter than for the consecutive times. The blocking period for the personal blacklist depends on a counter[26] c_{bl} that increases every time the author

[26]See *Personal Lists* in Subsection 3.6.1.

is blocked[27]. This adaptive mechanism remembers past misbehaviors and increases the blocking time whenever the author is blocked again. It's like putting the author on probation, increasing the punishment with every additional violation.
In the current design, we calculate the blacklist timeout t_{bl} by

$$t_{bl} = c_{bl} \cdot 28 days \tag{3.9}$$

which is quite high compared to the other timeouts in this section. We set it that high because objectively bad rated content leading to blacklist nominations is clearly unwanted. This is much worse than content of bad quality or not according to our taste which is just not what we expected. An author on the local blacklist is blocked for as long as there are enough recommendations available. Whenever the recommendations become insufficient, the author is blocked for 7 more days which equals t_{sugg} above.

Blocking Time: Whenever the combined rating reaches a certain negative threshold, the corresponding author is blocked. The blocking time t_{block} depends on the absolute value of the combined rating. In this design, we calculated it by multiplying the rounded absolute value of the combined rating cr_{author} with 14 days (2 weeks), i.e.

$$t_{block} = round(cr_{author}) \cdot 14 days. \tag{3.10}$$

When assuming a blocking threshold of -2, every author is initially blocked at least for 4 weeks. If bad content is received after this blocking time, the rating decreases and gets blocked immediately for a longer time. The content will not be blocked when rated positively, even if it still remains below t_{block}.

3.7. Spam Control

There are two types of spamming are possible, content (episode) spamming and channel spamming. The former includes traditional spamming of content as known from e-mail spam, the latter is specific to the discovery channel of PodNet and comprises the generation of many channels to which users could subscribe. In this section, we will differentiate between those two.

3.7.1. Content Spamming

In order to limit and impede the spreading of spam and other unwanted content, two main techniques are used. The first one is blacklisting whereas the second and more proactive one, is rate limited publication.

[27]Optionally one could subtract the subjective rating from the counter before multiplying it in order to block people with good ratings for less time.

3.7.1.1. Blacklisting

The blacklisting mechanism comprises several blacklists (see Figure 3.3) used in different ways. There is a personal blacklist, containing the personal opinions, then there is a local blacklist which reflects the opinion of ones surrounding, a channel blacklist is used in order to empower channel moderators to ban certain authors and optionally a global blacklist could be used if a central authority is available.

3.7.1.2. Rate Limiting

The rate limitation regulates the download of content and allows only a limited amount of downloads per day and author. This avoids spam floods and it gives the users some time to review and rate the content. There are two reasons why we don't limit the rate at the sender. Firstly, attackers modifying the code[28] could easily bypass such security measures and secondly, a supplier does not know how much content we already received from an author or the rate limitation we currently have.
The actual rate per author starts with one content per day and is dependent on the authors rating on the personal rating list. Note that we base the rate limit only on the personal rating so that no attacker can increase the ratings of his/her sybil users in order to increase the publication limit. If the personal rating r_p of an author is greater than 0, the maximal amount of content one might get from that author is calculated the following way:

$$2^{10 \cdot r_p} \tag{3.11}$$

If no personal rating is available, a rate of 1/day is set as default. This produces an exponential increase of the publication rate limit.
Making the rate limitation dependent on the personal rating has the positive side effect that users wanting to receive more content from an author, have to rate him/her and therefore, the rate limitation serves as an incentive for rating the content. However, assuming an attacker is generating many different user IDs, he/she could still flood a channel even if the rate limitation is applied. Therefore, it may be necessary to add an additional download limitation of e.g. maximum two contents per day and channel from unknown[29] and untrusted authors, i.e. authors not having CA certified credentials, in order to reduce the effect of potential spam attacks.

3.7.2. Channel Spamming

A spammer could also deliver messages in channel titles or flood another user with a huge amount of generated channels so that the user can hardly find any usable channel to subscribe. We currently did not do any changes to the discovery channel and include all local channels without

[28] Compare *Cheating Attacks* in Section 3.2.
[29] Unknown means: no credentials available.

checking any legitimacy or perform a spam protection. The reason why we do so is the fact that we generate separate channel credentials and allow an owner to anonymously create a channel as explained in Subsection 3.4.1. The main reason for this was mainly the possibility that a channel owner can pass the ownership over to somebody else if he/she would leave the network or wants to quit channel administration.
If one would generate the channels with the author credentials, one could use an additional blacklist for channel owners tackling channel spamming. But this would lead to an additional overhead which may not be wanted at all.
Alternatively, one could design the discovery channel differently as explained in Subsection 5.6. The main changes would possibly comprise the introduction of channel aging and timeouts for received channels suggestions and/or the exchange of only locally subscribed channels. Additionally, channel quality or availability information could help to improve the relevance of the shown channels in the discovery channel.

3.8. Conclusion of Design

In this section, we will shortly summarize and conclude the most important parts from the design and present potential improvements for future work.

3.8.1. Summary

We will give here a short summary over the key point in the design.

Authentication:

- Every user generates its own identity including a public/private key pair. The unique identity inside PodNet is ensured by using the hash of the public key as identifier.
- Other user can ensure a user's identity by challenging him or her for the private key.
- Friend ties increase trust in specific users and are performed via secure pairing. As proof of the friendship, friends sign each other's credentials.
- Optionally, but not yet implemented, partially offline authorities could also sign the user's credentials by a mechanism that ensures human identity.

Authorization:

- *Open*, *restricted* and *closed* channels provide containers for different needs. Everybody is allowed to publish in and download from open channels. In restricted channels only a limited group of people can write but everybody can see the content. Closed channels are defined for small groups where information is private and is thus encrypted.

- Corresponding to the three channels, we define three user roles, i.e. *owner*, *moderator*, *member* and all others being regular users.
- The owner has created the channel and is allowed to nominate moderators (*Moderator List*) which administer the channel. Owner and moderators are allowed to include members (*Member List*) in the channel and give them certain read and/or write rights. In particular, in restricted channels, they define which members are allowed to publish content whereas in closed channels, they define which users can view and/or publish content.
- Global blacklists are generated by central authorities and are exchanged between all peers (if needed) just after connection.
- Channel blacklists are generated from channel owners and moderators and are only valid within the same channel. These lists are exchanged among all subscribed users together with the channel meta data.
- Every content is signed by its author and the signature as well as the author's certificate is included in the episode meta data. Every user can check integrity upon reception.

Trust:

- Trust is automatically assigned by familiar and community detection. The criterion to be included bases on time and similarity of the environment.
- Additionally, stronger trust can be assigned by consciously constructing *Friends* relations. The exchange of friends list can help automatically find the two-hop friends and include them in the *Friend Circles*.

Reputation:

- The author reputation is built by a two-level rating assessing the legitimacy (*objective rating*) of and the satisfaction (*subjective rating*) with the content.
- Every user can rate downloaded content by himself/herself. Non-legitimate contents, i.e. spam, fakes, illegal/objectionable content, are usually blocked directly whereas bad rated content influences the spam control mechanism (see below). Additionally, if several contents from the same author get a bad rating from the user, it will be blocked temporarily as well.
- The subjective rating is based on a binary values. The cumulation of ratings can still lead to a multi-level rating.
- The rating information (subjective and objective) is exchanged among users which are subscribed to the same channel before the exchange of any content and even if no other data is transferred.

- In order to reduce the influence of liars, the recommendations (received ratings) are weighted with the users' trust value. Received objective ratings have to exceed a certain *suggestion threshold* before they have an effect.

- The received objective ratings are currently processed in order to block misbehaving authors after exceeding the suggestion threshold. The exchanged subjective ratings are currently only used to calculate the *Blocking Time* but could gain importance in future applications, as for example to regulate the amount of received content (see next Subsection).

- The duration of blocking an author increases with the number of his/her misbehaviors.

Spam Control:

- The spam control mechanism targets publication rate of users inside a channel. It starts with 1 content per day and increases with a higher subjective rating.

- In order to prevent sybil users to switch identities for spamming, only a maximum amount of content from unknown authors is allowed.

3.8.2. Future Work

In this subsection, we list potential extensions and improvements to the current design. Most of them are already mentioned as *optionally* in the text.

CA Registration:

- The Registration process by a partially offline authority is currently not implemented. However, in a hybrid network like PodNet, it would be straightforward to perform such an action by PodNet gateways.

- Performing the entire registration process by SMS/MMS would make the registration quite location independent in many industrialized countries. Since PodNet is assumed to run on smart phones, that may not be a huge restriction.

- An authority could check additional properties as for example the legal age of a user for parental control. Channel owners may require the users to fulfill certain criteria and require the reception of certain CA signatures before the user is allowed to participate in and download from the channel (i.e. channel policies).

- Authorities may advertise the registration process in their neighborhood by using the service mask field in the discovery message.

Channels and Content:

- The publication time included in the content (episode) meta data is currently not used. In future applications it may enable the download of new content or limit the dissemination of old information.
- The description of channels and content is currently made via human readable text. Future application could profit from extracting keywords from this description or even replace the description completely by tags. Keywords would enable channel/content search and the comparison of similar channels which could also be helpful for reputation (See below).
- Channel Policies could enable the channel owner to define channel specific rules, such as a sender rate limitation which is fixed and for everybody the same, or requiring users to obtain an authority certificate stating several specific properties (see above).
- Introduction of a revocation list: A list inside a channel that contains all episode IDs that should be deleted by the users in the channel. This list would enable users to delete specific content without blocking the author.

Rating:

- By exploiting the two level rating, a user could regulate the amount of content he/she gets in a more sophisticated way than only with the spam control mechanism. If not much content is available, he/she can neglect the subjective rating and only look for legitimate content to get everything that is legitimate. On the other hand, if a lot of content is available, he/she can regulate the amount of download content by additionally requiring a certain positive rating.
- It may be interesting to investigate whether weight refinements, i.e. giving more weight to people who rate content similarly, would improve accuracy.
- One could substract the subjective rating from the blacklist counter before calculating the blacklisting timeout in order to block people with very good ratings (besides the blocked content) for less time.

Distribution:

- In order to save resources, a supplier could decide to spread content only for a certain time or with a certain probability. The usage of incentives to reward suppliers may motivate people to provide content and ensures fairness in the distribution process.
- If some downloaded data is corrupted or malicious, the information about the supplier could be remembered and appropriate countermeasures can be taken in the future. Although the supplier cannot be held accountable for the content itself, he/she should have checked the integrity of the downloaded content before supplying it to others. Since an authentication is performed before every download, the supplier cannot pretend to be someone else.

- When selecting a peer as supplier we only consider his/her trust value. Other factors, like providing special services[30] or having a good author rating[31] might be taken into consideration. Additionally, one could prefer suppliers who provided good content in the past.
- In particular in closed channels, user may not want to reveal any informations to unauthorized persons. Therefore, an additional authentication of the downloader (additional to the supplier authentication) may be reasonable.

[30]See Subsection 3.3.3.

[31]In case the peer is an author.

4. Simulation

In this chapter all the aspects concerning the simulations are described, starting with the *Scenarios* which highlight the simulated behaviors. Then, the data sets which were used in order to perform the simulations are presented. Furthermore, the setup of the simulations is defined. Finally, the results of the simulated behaviors are shown.

4.1. Scenarios

In our simulations we focus on scenarios operating in the open channels. Limiting the dissemination of bad content in closed or restricted channels is less difficult since only a limited group of well-known people is allowed to publish content. The problem arises when trying to control content dissemination in open channels where everybody can publish.
Every scenario requires a model of the user's behavior, specially when it comes to rating content. Modeling a user's rating behavior is a difficult task since it comprises taste and social aspects which need to be evaluated in a real-world deployments. Therefore, we only simulate objective rating[1] resulting in blacklists. The exchange of global blacklists made by a central authority is not simulated at all since it is an add-on based on traditional reactive spam protection used e.g. in e-mail systems. We allow its usage but do not simulate its efficiency because they are already well known and lead to a spam protection at least as effective as with moderator blacklists. The following five behaviors are simulated.

Non-cooperating Nodes: Non-cooperating nodes behave in such a way that they download content but do not provide it to other nodes. There are two reasons why we simulated the effect of non-cooperating nodes: Firstly, to test the resilience of the network against freeriders which do not cooperate on purpose. Secondly, to study the effect of different nodes not participating in the distribution of certain content because of their particular taste.

Moderator Blacklisting: Moderators are regular users which have the capacity to put misbehaving nodes on a channel blacklist as explained in Subsection 3.4.3. In our model they detect spam upon content reception and generate blacklists which are only valid within the channel. We assume that they are known by all users so that they can accept the blacklists immediately

[1]See Section 3.6 for objective rating.

upon reception[2]. In order to increase the dissemination speed, the received channel blacklists themselves are exchanged among all users of a channel before sending any content.

Personal Blacklisting: Assuming a user not wanting to rely on other persons' opinions at all, he/she handles his/her own blacklist based on his/her own experience. Every user detects malicious messages with a certain probability but does not share that information with other nodes. If content gets blocked, it is not advertised anymore to other users and no other content from the same author is downloaded.

Locally Shared Blacklist: As with personal blacklists, every node detects spam messages with a certain probability but additionally exchanges this information with all nodes it communicates with. In order to avoid exponential spreading, only direct information, i.e. information based on downloaded content, is exchanged. Indirect information, i.e. suggestions received from other users, may only influence the download decision of new content locally. Because open channels are susceptible to liars, the received recommendations cannot be immediately accepted and suggestion thresholds[3] are used. We will observe the effectiveness of sharing blacklist information with others compared to keeping it secret as with personal blacklists.

Send Rate Limitation: Because blacklists are reactive and slow, an author that just wants to flood the network with spam messages would not be prevented from conducting his/her plan. We therefore suggested in Section 3.7 the usage of a regulated send rate which limits the amount of messages received by the same author. We combined this simulation with local blacklisting and simulated a constant send rate limitation of 1/day[4].

Socially Weighted Recommendations: Assigning a higher weight to recommendations from community members may improve the susceptibility to occasional attackers as explained in Subsection 3.5.1 by allowing the adjustment of the suggestion threshold for the local blacklist. We will compare the community weighted recommendations with results from regular local blacklisting.

Note that all simulations showing the spreading performance of exchanged blacklist, in particular the moderator blacklists and all local blacklists, can be considered as worst case scenarios because we assume unlimited bandwidth resulting in an immediate exchange of every user's blacklist and contents.
However, in reality, users may not be connected long enough to exchange all their information. Whereas many blacklist recommendations may fit in one packet, a single data content itself may require several packets depending on the content size. Since blacklists are exchanged before

[2]As explained in Subsection 3.4.3, channel blacklists would be signed allowing only the moderators to change any information on it.
[3]Defines the minimum number of recommendations needed before accepting the information.
[4]Send rate increase based on the rating is neglected due to simplicity reasons.

data transmission, the blacklist information would spread much faster, particularly in mobile, opportunistic networks, where short connection periods are common and the transmission of large data is hindered.

4.2. Datasets

The simulations are based on real world mobility traces as well as synthetic models. We first used the MIT reality traces[5] because it is the largest publicly available data set. The set consists of data from 96 distinct persons[6] which was collected over a period of several months. The spreading quality of the individual weeks varies a lot because of the large collection period comprising all semester breaks, holidays, Christmas breaks and semester starts which are all different from the regular semester. In a second simulation round, we therefore classified the weeks into good, bad and average weeks. Details about the classification process can be found in Appendix B.2. However, the connection density was still quite low even in good weeks because during the data collection, the bluetooth scanning interval was reduced to one scan every five minutes in order to increase the standby time of the mobile phones[7].
We repeated some simulations with the Haggle iMotes Infocom traces due to a higher connection density[8]. However, the Haggle dataset is much smaller and comprises only 41 persons over a period of three days which gives only limited insights into human interactions. Both datasets were obtained from the Crawdad repository[9].

As reference, we repeated some simulations using generated traces from a Random Waypoint model (RWP) consisting of 100 nodes moving at a speed of $0.5\frac{m}{s}$ and $1.5\frac{m}{s}$, a topology of $1000m \times 1000m$, a transmit range of 10m. We consider nodes to be in contact if it lasts at least 10s. Additionally, we used a modeled environment of Helsinki[10] which is more sophisticated than pure Random Waypoint since it is map based and contains points of interest. Both traces were generated over a period of 2 weeks.

4.3. Simulation Setup

In general, we assume that two nodes exchange and synchronize all their contents when they meet each other regardless of their connection duration (except for RWP and Helsinki). We assume that all users are subscribed to the same open channel and all information is exchanged within this channel. Every simulation is repeated several times by selecting a different random source node in every round and averaging over all the simulations to obtain the final simulation result.

[5]http://reality.media.mit.edu/
[6]Note that one person does not have any contacts with any other persons and is therefore excluded.
[7]http://www.crawdad.org/meta.php?name=mit/reality
[8]The resolution is also slightly better since they have a two minutes scanning interval.
[9]http://crawdad.cs.dartmouth.edu/
[10]This model consists of 126 nodes and came along with *'The One'* Simulator which we used for the simulations as explained later.

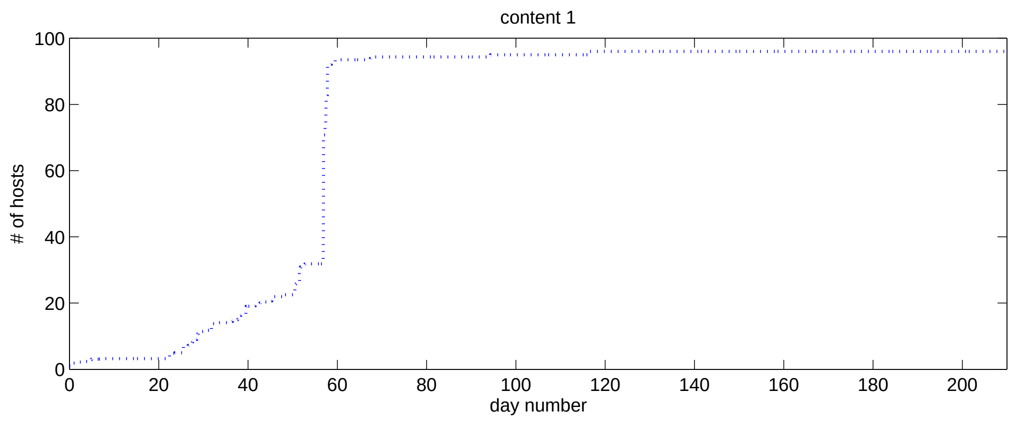


Figure 4.1.: Normal Content Spreading in MIT Set: *One content is generated at day 0 without applying any spreading restrictions. Huge increase around day 60 correspond to semester start at MIT comprising much more connections.*

We avoided simulating the content generation with all possible nodes due to time limitations. However, the simulations seem stable as results for different randomly chosen nodes look very similar. More information about the randomly selected sources can be found in the Appendix B.1.

The traces were replayed with *'The One'* simulator[11], an opportunistic network simulator developed at University of Helsinki, which we extended so that all nodes could process the received messages individually and perform actions based on both the message and their internal state. A detailed explanation of the extension can be found in Appendix A.

Because of the different duration of the trace data we simulated the message generation differently. When using the MIT dataset, every source generates one content per week on Monday morning at midnight. When using RWP and Helsinki, one content is generated per day and when using the Haggle traces, we simulated the content generation every 12 hours.

4.4. Simulation Results

In this section the results of simulating the different scenarios introduced in Section 4.1 are presented. They consist of the effect of non-cooperating nodes, the advantages of blacklists created by moderators and the performance comparison of personal versus the locally shared blacklist. Additionally, the importance of limiting the send rate as well as the positive effect of exploiting social ties are included as well.

In Figure 4.1 the normal spreading of a content in the unclassified MIT data set is shown. We see a huge increase around day 60 which corresponds to the semester start at MIT[12]. All the content sent before the semester start will only spread poorly because of the inactivity of most individuals. The semester start itself comprises the most connections because all people are

[11]http://www.netlab.tkk.fi/tutkimus/dtn/theone/

[12]See Appendix B.1 for more information.

present at that time. If assuming the continuous generation of one content per week, this will result for most people in a burst of 8 content at semester start where they first see people providing that content. Because of the quality differences of the individual weeks, we classified the data set into three sets of good, bad and average weeks and replayed some weeks in order to get data set sizes of 30 weeks. Details about the classification can be found in Appendix B.2. The simulation results obtained with good weeks look similar than with unclassified weeks starting at week 9 (around semester start). For a better visibility, we don't show the unclassified plots in this section but they can be found in Appendix C.

All spam simulations shown in this section[13] were performed with a malicious content recognition probability of 10% on average[14]. We simulate a pessimistic scenario with a low spam recognition because we assume that users may not vote that frequently. When assuming a higher recognition of malicious content, the results would be much better as some simulation results in Appendix C show. We cannot expect from every user to detect malicious content at a very high rate since recogntion depends on the nature of the content, i.e. classical spam would be recognized with a higher probability than malware or wrong information.

4.4.1. Non-Cooperating Nodes

The spreading performance in the presence of non-cooperating nodes is simulated using different traces, either realistic or synthetic, for parameters ranging from full cooperation to nearly total non-cooperation as summarized in Table 4.1. These results enable the analysis of the influence of nodes which do not participate in the dissemination of content, either deliberately or for a lack of interest.

Simulations with Non-Cooperating Nodes				
	Unclassified MIT	**Classified MIT**	**Haggle**	**Synthetic Models**
Parameters	23 author seeds 5 non-coop. seeds 23×5 *sim rounds:* 1 author 1 content/week 0-90% non-coop.	23 author seeds 11 non-coop. seeds 23×11 *sim rounds:* 1 author 1 content/week 0-90% non-coop.	25 author seeds 7 non-coop. seeds 25×7 *sim rounds:* 1 author 1 content/12h 0-90% non-coop.	11 author seeds 7 non-coop. seeds 11×7 *sim rounds:* 1 author 1 content/24h 0-90% non-coop.
Figures	Fig. C.1	Good: Fig. 4.2 Average: Fig. C.3 Bad: Fig. C.2	Fig. C.4	Random: Fig. C.5 Helsinki: Fig. C.6

Table 4.1.: Overview of all Simulation Parameters with Non-Cooperating Nodes.

In Figure 4.2, the average spreading performance of good classified MIT weeks is shown over a time period of four weeks. The percentage of non-cooperation ranges from 0% to 90%. The graph shows a relative exponential performance drop with linear increase of non-cooperating

[13]Except the simulations with moderators where a content recognition of 100% is assumed and all users accept moderator suggestions upon reception.

[14]We defined a predisposition to rate between 0% and 100% for every user in order to model some users which rate very often whereas others only seldomly do so. Every user has thus a spam recognition between 0% and 20% which lead to an average spam recognition of 10%.

nodes. The reason may be the breaking of some weak links that connect different social groups at higher levels of non-cooperation. Up to 30% of non-cooperation, the performance is almost exactly the same and even with a rate as high as 90%, still approximately two thirds of the people would receive the content[15].

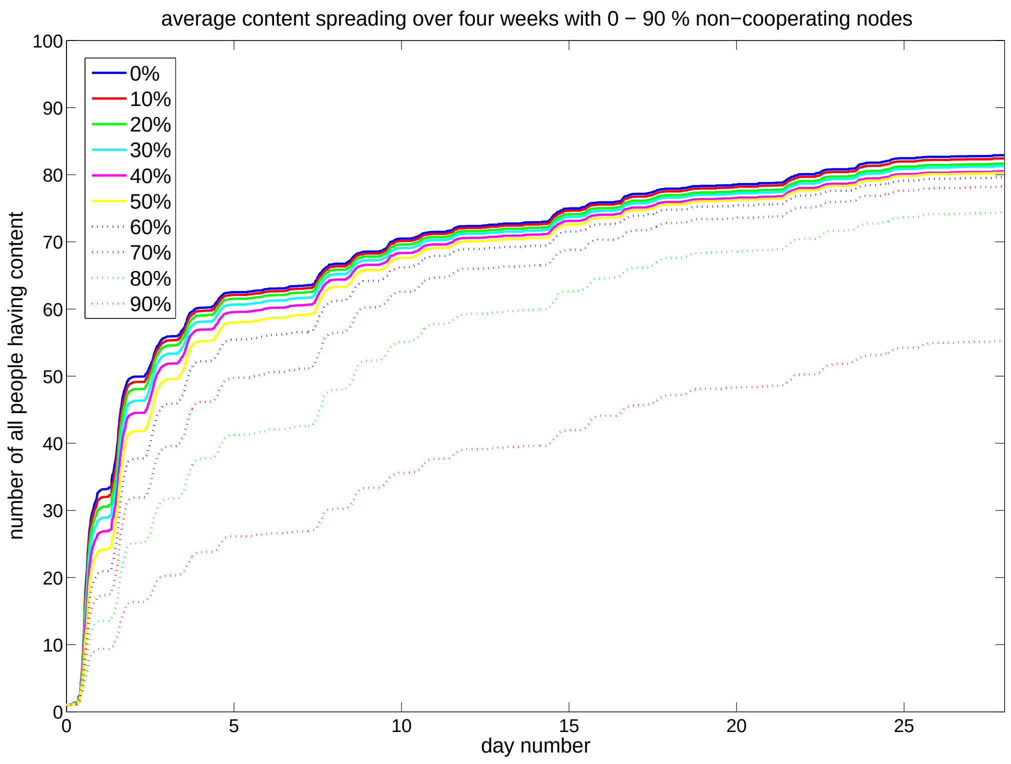


Figure 4.2.: Non-Cooperation in Good Classified MIT Weeks: *Average spreading performance of one content generated at day 0, observed over a period of four weeks. Linear increase of non-cooperation leads to exponential decrease in spreading performance.*

The simulation results for other data sets, i.e. bad, average and unclassified MIT weeks, Haggle traces as well as Random Waypoint and Helsinki models can be found in Appendix C.1. Depending on the mobility environment (e.g. campus, conference or synthetic), the timescale of dissemination varies. Table 4.2 shows the average spreading performance[16] of all *real world* traces. In good classified MIT weeks content needs four weeks until reaching 86% of all people whereas in the Haggle data set, it needs only 24 hours to reach the same level. Figure 4.3 compares the performance of content spreading with varying number of non-cooperating nodes (for 0% to 90%) for different traces. Because of the different connection density, we show con-

[15]Compared to the people that would receive it with fully cooperating nodes.
[16]Assuming fully cooperating nodes.

tent dissemination over four weeks for all MIT traces whereas over 24 hours for all other data sets.

Dataset	Percentage of total nodes
MIT Unclustered	68.00%
MIT Good	86.37%
MIT Bad	41.08%
MIT Average	64.15%
Haggle	86.85%

Table 4.2.: Average Spreading Performance: *Percentage of people from entire dataset receiving the content in four weeks (MIT) or 24 hours (Haggle). The numbers are relative to the entire data set size of 96 people (MIT) and 41 (Haggle).*

The susceptibility to non-cooperation depends on the intensity of the environment which depends on cycles (semester, breaks). However, the susceptibility to non-cooperation also depends on the quality of the environment as Figure 4.3 shows. In good classified MIT weeks, even if 90% of all nodes do not cooperate, more than $\sim$ 66% of the original nodes would receive the content in four weeks whereas in a bad environment, only $\sim$ 37% would get it. This result seems obvious because in a bad environment, less connections exist and only 41% of all people[17] would get the content assuming full cooperation of all nodes. Therefore, the non-cooperation of individual links affects the content dissemination to a higher extend[18].
In a high density environment like Haggle, where almost all nodes see each other within 24 hours, the spreading performance is only slightly decreased up to a very high percentage of non-cooperation because still many connection links exist. These results are consistent with the findings from [33].

[17]Compare Table 4.2.
[18]The details can be found in Appendix C.1.

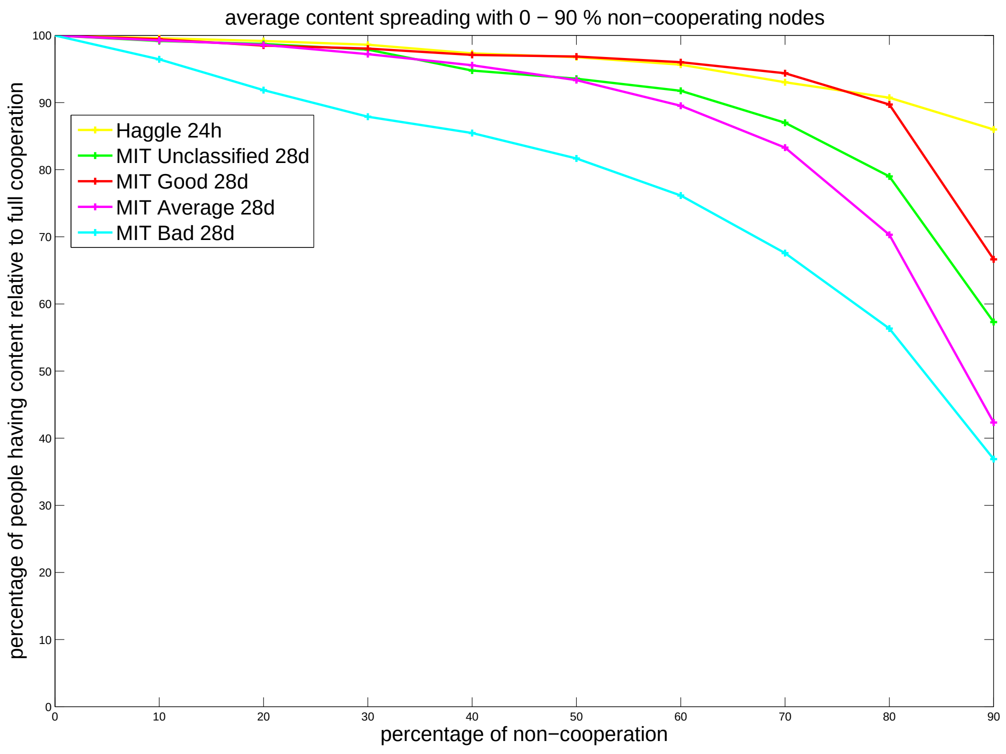


Figure 4.3.: Non-Cooperation compared to Full-Cooperation: *Different levels of non-cooperation compared to full cooperation for the Haggle and MIT traces. The time scale varies between 24 hours (Haggle) and four weeks (MIT). The distribution performance decreases faster for traces comprising fewer connections.*

4.4.2. Moderator Blacklisting

In this subsection the results of simulating the spreading performance of spam when using the moderator blacklist is presented. All the performed simulations are consolidated in Table 4.3. Through the results the efficiency of the global and the channel blacklist described in Subsection 3.4.3 can be assessed.

Simulations with Moderator Blacklists			
	Unclassified MIT	**Classified MIT**	**Haggle**
Parameters	23 spammer seeds 5 moderator seeds 23 × 5 *sim rounds:* 1 spammer 1 content/week 1 moderator p_{req}: 100%	23 spammer seeds 31 moderator seeds 23 × 31 *sim rounds:* 1 spammer 1 content/week 1 moderator p_{req}: 100%	11 spammer seeds 23 moderator seeds 11 × 23 *sim rounds:* 1 spammer 1 content/12h 1 moderator p_{req}: 100%
Figures	Fig. C.7	Good: Fig. 4.4 Average: Fig. C.8 Bad: Fig. C.9	Fig. C.10

Table 4.3.: Overview of all Simulation Parameters with Moderator Blacklists.

The channel blacklist is generated by moderators and exchanged among all users of a channel. Every user accepts and uses the channel blacklist just after reception and before data transmission. In Figure 4.4, the average spreading performance of content generated by a randomly selected spammer is shown in the presence of a randomly selected moderator as specified in Table 4.3. Whenever the moderator receives content, he or she recognizes it with a probability p_{req} of 100% and starts exchaning the channel blacklist. The randomly selected spammer generates a new content every week.

The x-axis in Figure 4.4 shows the number of days since the first spam generation. The y-axis shows the number of people that have received the content. In particular, the blue dotted line shows the number of hosts that would have received the content if no countermeasures were taken, the solid blue line shows the hosts that do provide the content to others after reception and the red line shows the users that have received the content but have blocked it. Thus, the solid blue line together with the red line shows the total number of people that have received the content and the difference between (red and blue) line to the dotted line shows the nodes that have blocked the content without having received it.

Although the moderator receives the content the same day of generation and recognizes it with a probability of 100%, there is still a fast increase of people providing the first content in the uppermost graph in Figure 4.4. At its maximum 79% of all people that have seen the content have also received it and almost 62% of them do provide it to others. In a good environment, malicious content spreads very fast but on the other hand, as soon as the moderator has detected the content, the blacklist will spread fast as well. Therefore, we see a high sharp peak of people providing the content in Figure 4.4.

The other graphs in Figure 4.4 show the content spreading for the next three contents generated

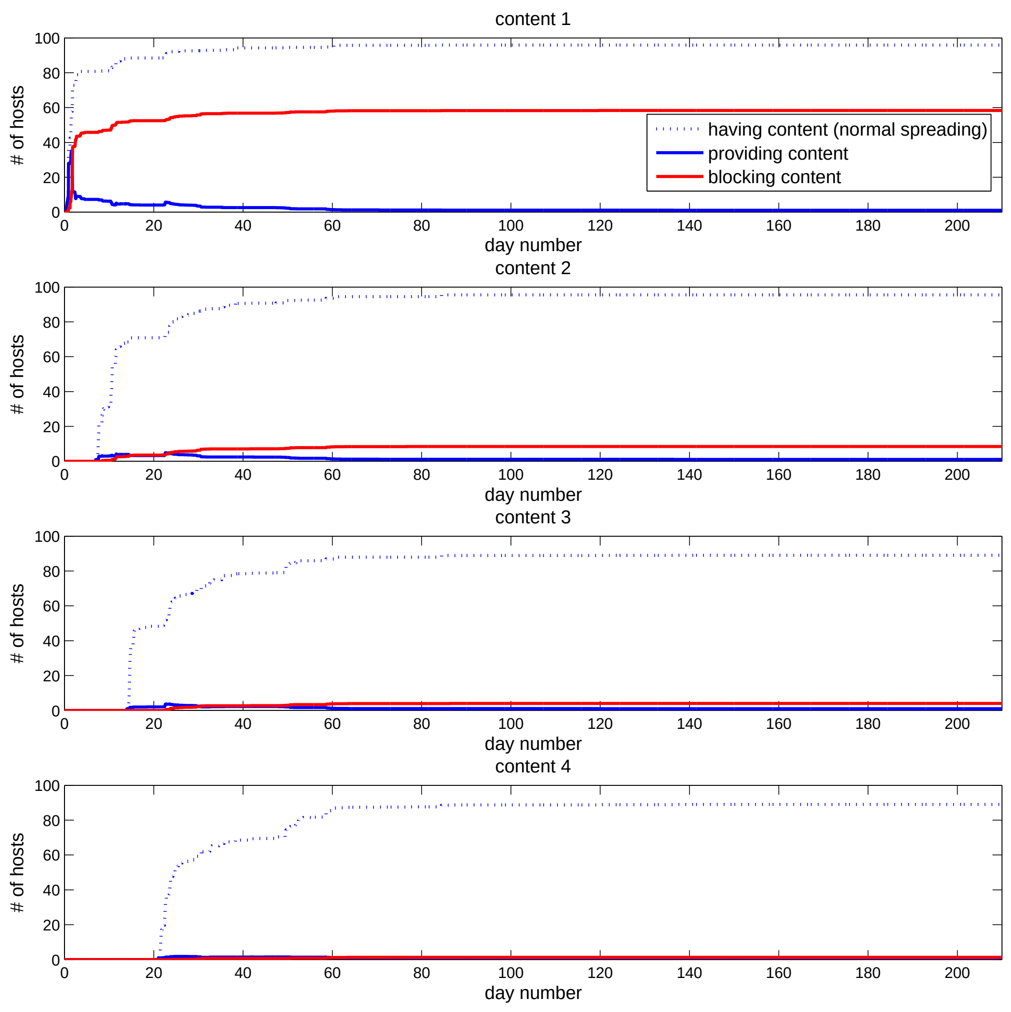


Figure 4.4.: Moderator Blacklisting in Good Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. One moderator detects spam upon reception with recognition probability of 100% and exchanges that information with others which immediately accept the blacklist information. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

in the following three weeks respectively. We see that the dissemination of these is neglectable since most users are already blocking the content after the first week.

We repeated the simulations in bad and average classified MIT weeks in order to see the spreading effects in environments with worse distribution conditions. The figures can be found in Appendix C.2 together with the simulation results of the unclassified MIT and Haggle data set. The spreading performance of the first content generated in the first week in the presence of only one moderator is summarized in Table 4.4.
In a good and average environment, the moderator will get the content already the first day whereas in a bad environment, it takes 9 days on average until it reaches the moderator. The weeks in Table 4.4 are all relative to the moderator's recognition of the content.

	Good		Average		Bad	
time	having	providing	having	providing	having	providing
1 week	65.79%	8.63%	75.39%	40.04%	83.32%	71.06%
2 weeks	63.99%	4.78%	67.11%	27.33%	82.77%	71.57%
4 weeks	64.07%	4.22%	63.10%	13.37%	77.01%	52.73%
8 weeks	62.83%	1.96%	62.60%	6.80%	62.23%	14.98%
16 weeks	62.01%	1.18%	61.12%	1.39%	57.05%	3.78%

Table 4.4.: Moderator Blacklists: *The table lists the percentages of people providing and having the first content in good, average and bad classified MIT weeks. The week times are all relative to the spam recognition of the (only one) moderator. The percentages are obtained with respect to the people receiving the content with normal spreading.*

As mentioned above, the percentage of people providing the first content in a good environment forms a sharp maximum on the same day of publication. However, one week after the publication, the malicious content is well known and less than 10% of the people that have seen the content also provide it to others. The fewer connections an environment has, the slower the malicious content disclosure. In a bad environment, people will not start to blacklist the author immediately and the percentage of people providing the content will still increase until two weeks after recognition of the content. The reason is the bad spreading of the moderator's blacklist.

4.4.3. Reputation Spreading

In the last subsection, we've seen the spreading performance when using a channel blacklist generated by a moderator. All users immediately accepted the blacklist but did not perform their own malicious content recognition. In this subsection, we will observe the spreading performance when relying on objective ratings. The measures influencing objective reputation consist of two parts, the personal and the local blacklisting. In order to show the effectiveness of both approaches, we evaluate them separately. The results are presented in the this section.

4.4.3.1. Personal Blacklisting

As explained in Section 3.6.1, every device handles its own personal blacklist based on user input.

Simulations with Personal Blacklist				
	Unclassified MIT	**Classified MIT**	**Haggle**	**Synthetic Models**
Parameters	23 spammer seeds *in each round:* 1 spammer 1 content/week p_{rec}: 10%, 50%	23 spammer seeds *in each round:* 1 spammer 1 content/week p_{rec}: 10%, 50%	25 spammer seeds *in each round:* 1 spammer 1 content/12h p_{rec}: 10%, 20%	11 spammer seeds *in each round:* 1 spammer 1 content/24h p_{rec}: 10%, 20%
Figures	p_{rec} 10%: Fig. C.11 p_{rec} 50%: Fig. C.12	Good: p_{rec} 10%: Fig. 4.5 p_{rec} 50%: Fig. C.13 Average: p_{rec} 10%: Fig. C.14 p_{rec} 50%: Fig. C.15 Bad: p_{rec} 10%: Fig. C.16 p_{rec} 50%: Fig. C.17	Haggle: p_{rec} 10%: Fig. 4.9 p_{rec} 20%: Fig. C.36	Random Waypoint: p_{rec} 10%: Fig. C.25 p_{rec} 20%: Fig. C.26 Helsinki: p_{rec} 10%: Fig. C.27 p_{rec} 20%: Fig. C.28

Table 4.5.: Overview of all Simulation Parameters with Personal Blacklists.

	Good		Bad		Average	
time	having	providing	having	providing	having	providing
1 week	100%	90.36%	100%	94.74%	100%	92.60%
2 weeks	99.90%	84.86%	95.28%	87.89%	97.02%	84.34%
4 weeks	96.52%	73.97%	95.96%	83.41%	100%	78.14%
8 weeks	95.95%	59.95%	90.76%	65.74%	100%	67.66%
16 weeks	95.92%	39.88%	85.67%	41.07%	98.37%	40.31%

Table 4.6.: Personal Blacklists: *The table lists the percentages of people providing and having the first content in good, average and bad classified MIT weeks. The week times are all relative to day 0 and the percentages to the people having the content with normal spreading. The spam recognition is set to 10%.*

In the the following simulations, we assumed that every user classifies spam just at reception with a certain recognition probability p_{rec} based on his/her predisposition[19]. The spreading performance is calculated by averaging over randomly selected spammers, each generating one new content every week as specified in Table 4.5. In Figure 4.5, the content spreading of the first four contents generated in the first four weeks is shown for a p_{rec} of 10%. Because of the low recognition probability, almost all users get the first content. By receiving more content in the following weeks, the probability of spam recognition increases because of the multiple rating opportunity which may result in more users blocking the author. Content from a blocked author will not be advertised anymore to other users and succeeding contents of the same author will not be accepted anymore. However, since no information is exchanged, every user has to recognize bad content on his/her own and it is likely that bad content will never disappear so that new users would always get it. In Table 4.6 the spreading performance of all the good, bad and average environments is shown for a spam p_{rec} of 10%. We see that spam recognition in all three environments decreases similarly with time, i.e. the percentages of people providing the content would be similar. The reason for that behavior is the fact that they all do not take advantage in exchanging their informations. The spam protection is therefore reduced to the isolated spam recognition of the individual users which is not very efficient.

[19]See beginning of this section.

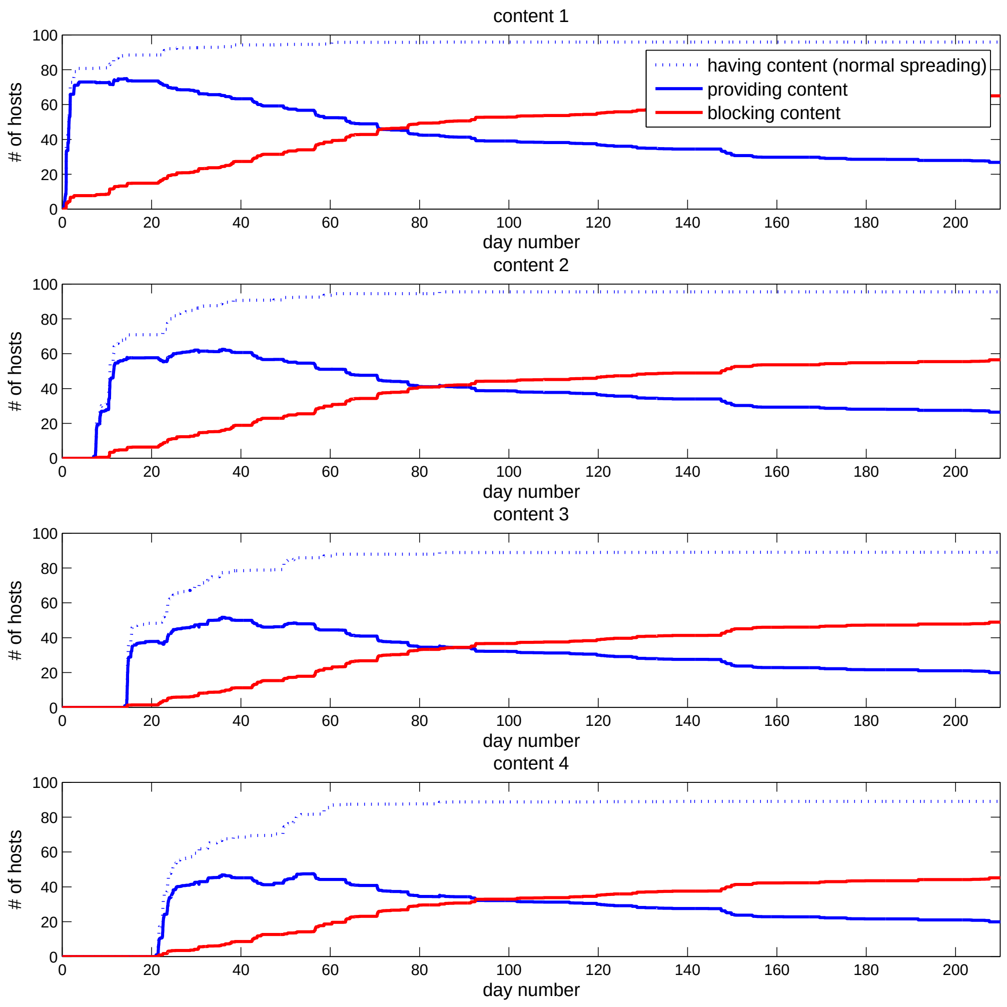


Figure 4.5.: Personal Blacklisting in Good Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All the users keep that information secret not notifying others. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

4.4.3.2. Local Blacklisting

In the previous subsection, we observed that keeping the information secret is not efficient at all. Now we share our opinion with others. Table 4.7 summarizes all simulations done with local blacklists.

Simulations with Local Blacklist				
	Unclassified MIT	**Classified MIT**	**Haggle**	**Synthetic Models**
Parameters	23 spammer seeds *in each round:* 1 spammer 1 content/week p_{rec} : 10% th_{sugg}: 10, 20	23 spammer seeds *in each round:* 1 spammer 1 content/week p_{rec} : 10% th_{sugg}: 10, 20	25 spammer seeds *in each round:* 1 spammer 1 content/12h p_{rec} : 10% th_{sugg}: 4, 8	11 spammer seeds *in each round:* 1 spammer 1 content/24h p_{rec} : 10% th_{sugg}: 10, 20
Figures	th_{sugg} 10: Fig. C.19 th_{sugg} 20: Fig. C.18	Good: th_{sugg} 10: Fig. C.22 th_{sugg} 20: Fig. 4.6 Average: th_{sugg} 10: Fig. C.23 th_{sugg} 20: Fig. C.20 Bad: th_{sugg} 10: Fig. C.24 th_{sugg} 20: Fig. C.21	Haggle: th_{sugg} 10 & 20: Fig. 4.9	Random Waypoint: th_{sugg} 10 & 20: Fig. C.25 Helsinki: th_{sugg} 10 & 20: Fig. C.27

Table 4.7.: Overview of all Simulation Parameters with Local Blacklist.

Exchanging information about malicious authors could improve the detection speed but in exchange the susceptibility to liars increases. As mentioned in Subsection 3.6.2, the single opinion of a user may not be trusted because he/she may be lying, but if a certain amount of recommendations[20] is received we, assume the information to be correct and trust the recommendation. The recommendations comprise only information from direct transactions and are exchanged before sending any other content.

Figure 4.6 shows the content spreading in good classified weeks for a recognition probability p_{rec} of 10% and a fixed suggestion threshold th_{sugg} of 20 recommendations. The threshold value corresponds to around 20% of all people in the data set which is very high, in particular when regarding the low p_{rec} of 10%. Thus, there is quite a large time delay until the threshold is reached and the measure takes effect.

Table 4.8 shows that after week 16, only 15% of the people are still providing the first content generated in the first week. This value is less than half the number that is observed with personal blacklisting. Other than with personal blacklists, we can see clear differences between the three environments in this case. While the content distribution in good environments is reduced by half compared to the personal blacklisting, it is only slightly decreased by around 12% and 14% in bad and average environments respectively. The reason for the difference in improvement is the fact that a fixed threshold of 20 suggestions was used which corresponds to different percentage

[20]Above the suggestion threshold.

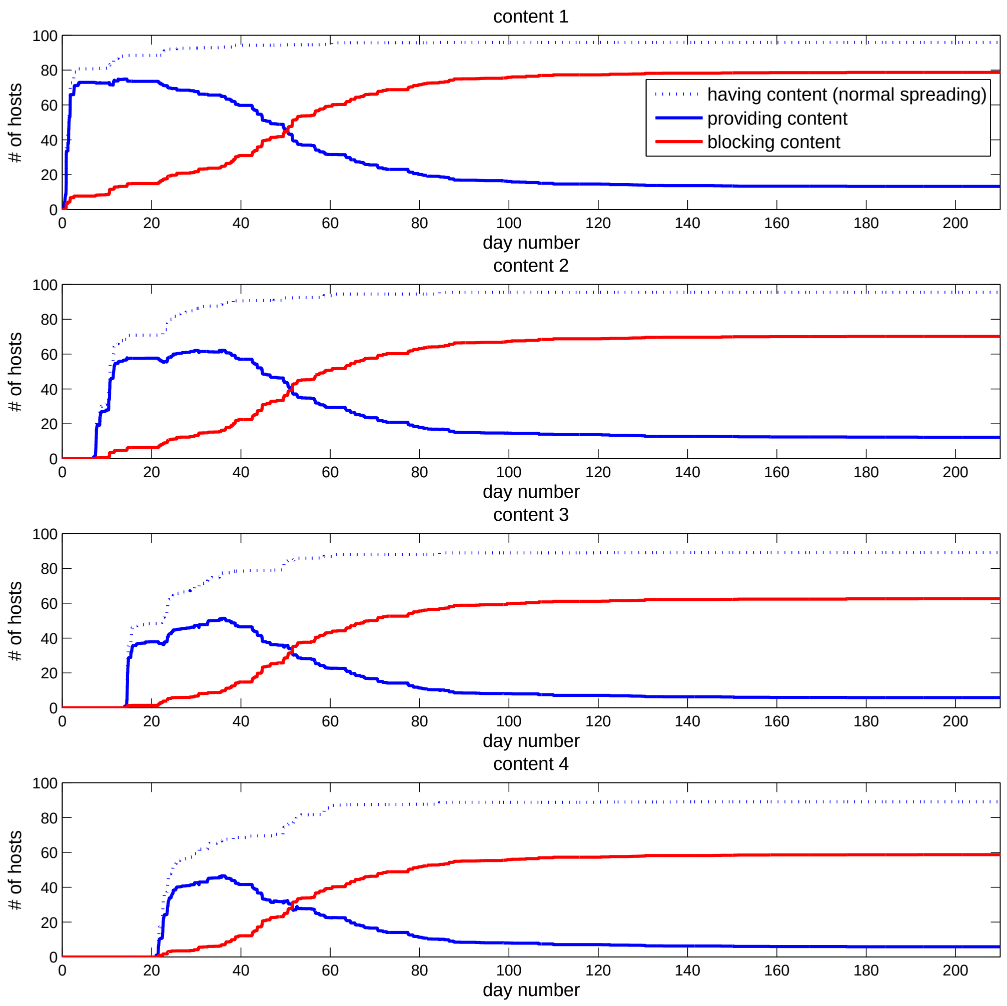


Figure 4.6.: Local Blacklisting in Good Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 20 opinions. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

	Good		Average		Bad	
time	having	providing	having	providing	having	providing
1 week	100%	90.36%	100%	92.60%	100%	94.74%
2 weeks	99.90%	84.86%	97.02%	84.34%	95.28%	87.98%
4 weeks	96.52%	73.97%	100%	78.14%	95.96%	83.41%
8 weeks	95.95%	38.76%	100%	67.10%	90.76%	65.74%
16 weeks	95.92%	15.29%	98.37%	34.64%	85.67%	35.96%

Table 4.8.: Local Blacklists: *The table lists the percentages of people providing and having the first content in good, average and bad classified MIT weeks. The week times are relative to day 0 and the percentages to the people having the content with normal spreading. The spam recognition is set to 10% and the suggestion threshold equals 20 opinions.*

time	Threshold 20	Threshold 10
1 week	90.36%	90.36%
2 weeks	84.86%	84.07%
4 weeks	73.97%	59.87%
8 weeks	38.76%	13.52%
16 weeks	15.29%	9.71%

Table 4.9.: Local Blacklist in Good Classified MIT Weeks: *The table shows the differences in people providing the first content when using a suggestion threshold of 20 or 10 opinions.*

of users present in the environment. Whereas in a good environment with many connections, the threshold is reached fast, it takes more time to reach the same amount of recommendations in a bad environment. Decreasing the recommendation threshold would increase the detection speed as shown by simulations in Appendix C.4.2 indicate. Table 4.9 compares the number of users providing content for two different values of the suggestion threshold. As long as the threshold is not reached, the simulations with both threshold values show the same performance. However, as the number of recommendations exceed the suggestion threshold, people start blocking the author and stop providing the content. In Table 4.9, a clear difference of people providing the content can be seen at week 4 which means that a suggestion threshold of 10 recommendations is reached between the second and forth week.

However, even when decreasing the suggestion threshold, there are still some users which provide the content to others because they either did not recognize it as spam or did not receive enough blacklist suggestions. This is not as bad as it seems because most people may have blocked the content and thus, the content may not start spreading again. A channel blacklist[21] made by moderators could complement local blacklists in order to achieve that an author is blocked by all users that participate in the channel.

4.4.4. Send Rate Limitation

Because of low a recognition probability or a high suggestion thresholds, users may receive spam before being able to block it. Therefore, a send rate limitation[22] is proposed to prevent users from completely flooding the network with new content. All simulations performed which show

[21]See Section 4.4.2.
[22]See Section 3.7 for more information.

it's effect listed in Table 4.10.

Simulations with Send Rate	
	Classified MIT
Parameters	23 spammer seeds *in every round:* 1 spammer 1 content/week p_{rec}: 10% th_{sugg}: 10, 20 send rate enabled
Figures	Good: th_{sugg} 10: Fig. C.33 th_{sugg} 20: Fig. 4.7 Average: th_{sugg} 10: Fig. C.32 th_{sugg} 20: Fig. C.29 Bad: th_{sugg} 10: Fig. C.33 th_{sugg} 20: Fig. C.30

Table 4.10.: Overview over all Simulations Parameters with Send Rate Limitation.

Therefore we generated a burst of 300 contents on the first week and observed the spreading using the send rate limitation.
We assume that a user allowing unlimited download will only rate an author once when he/she receives the content whereas other users using a send rate will rate each day they receive content. In Figure 4.7, we see the effect of a send rate compared to the unlimited download after the spam burst.

When allowing unlimited downloading, all the content would spread the same because every user would receive all the available content automatically at once without acknowledging or even noticing. After that, he/she would provide all the content to others and therefore help spammers to cheaply send content to many people. In the absence of a send rate, all contents would therefore arrive at day 1 whereas when using a send rate limitation of one content per day, the last content would be downloaded 300 days later.
We are aware of the fact that the simulated spreading performance shows an upper bound of the real performance since all ratings are done independently. It can be assumed that a user receiving 300 contents from one person at the same day would immediately block that author. However, a user receiving one spam content per day would certainly also start blocking that author long before receiving the last content.

There are mainly two advantages in downloading content from authors at a later stage. Firstly, the user may have more time to look at the content and block the author if needed. Secondly, information about the behavior of an author may already have spread in the network and a user can benefit from experiences of other users.

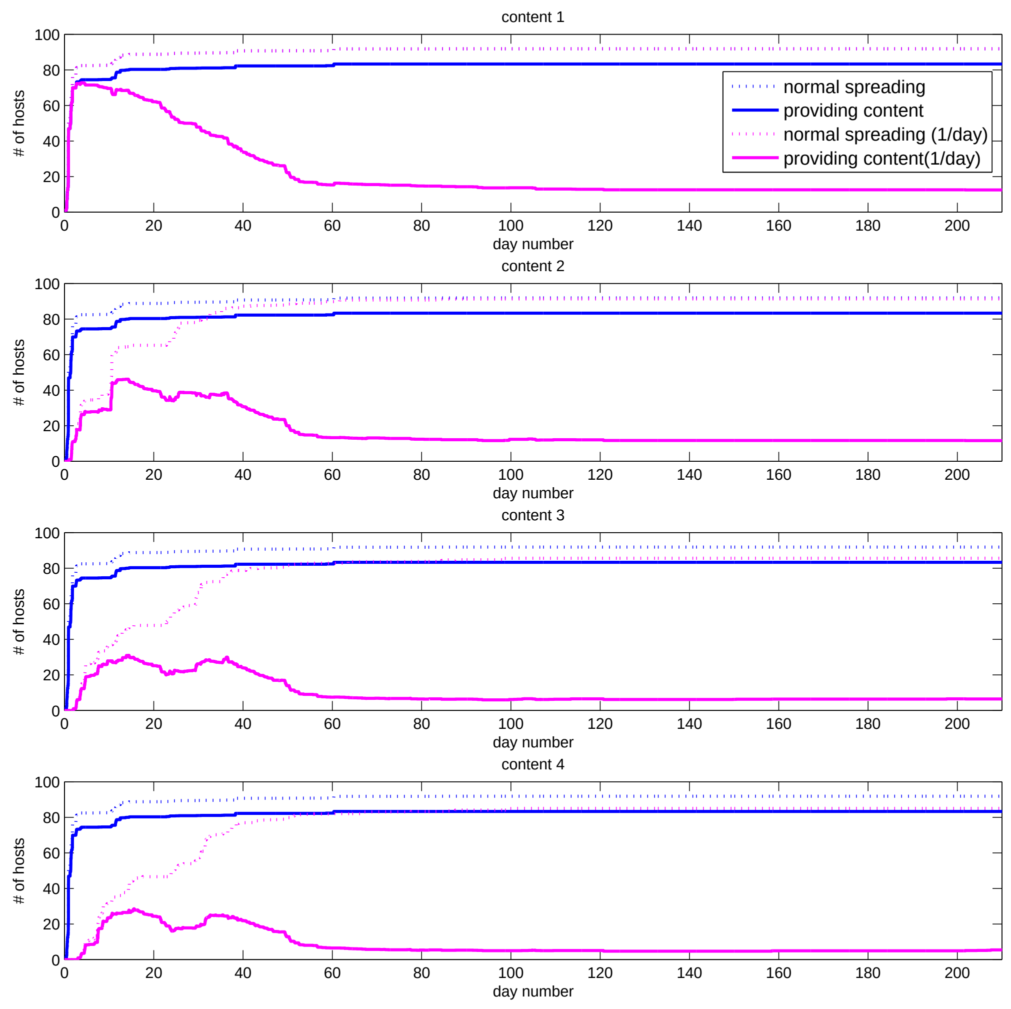


Figure 4.7.: Effect of Send Rate in Good Classified MIT Weeks: *Average spreading performance of 300 contents generated by a spammer at day 0. Every user detects spam with a recognition probability of 10% and exchanges that information with others. Received information is accepted after exceeding a suggestion threshold of 20 opinions. Without send rate of 1 content/day, all 300 contents are rated only once at first reception. When using the send rate, every user rates receives content every day after reception. The graph shows the people providing the first four contents of the burst.*

4.4.5. Comparison of Different Measures

In this subsection, we will combine our different approaches trying to compare their effectivity. All the performed comparing simulations can be found in Table 4.11.

Simulations Comparing Different Measures		
	Classified MIT	**Haggle**
Parameters	23 spammer seeds *in every round:* 1 spammer 1 content/day p_{rec}: 10% th_{sugg}:10%, 20%	25 spammer seeds *in every round:* 1 spammer 1 content/12h p_{rec}: 10% th_{sugg}: 10%, 20%
Figures	Good: Fig. 4.8 Average: Fig. C.35 Bad: Fig. C.34	Haggle: Fig. 4.9

Table 4.11.: Overview over all Simulation Comparing Different Measures.

In Figure 4.8, local blacklists with a suggestion threshold th_{sugg} of 10 and 20 recommendations and personal blacklists are combined with a send rate limitation of one content per day for a recognition probability p_{rec} of 10%. As Table 4.12 shows, the exchange of blacklist can indeed speed up the blocking process. Content that is blacklisted personally without exchanging the information will hardly die out assuming every user voting only once for each content. A system using a threshold of 20 opinions will behave the same than using personal blacklists until it reaches the threshold value and the content gets blocked. In the current simulations, this value was reached at day 17 as Table 4.13 shows. When decreasing the threshold value to 10, the content would get blocked earlier.

	Personal		Threshold 20		Threshold 10	
time	having	providing	having	providing	having	providing
1 week	100%	86.60%	100%	86.70%	100%	85.96%
2 weeks	99.12%	77.44%	99.12%	77.44%	99.12%	68.21%
4 weeks	99.22%	62.31%	99.22%	55.89%	99.22%	48.64%
8 weeks	99.28%	39.05%	99.28%	18.40%	99.04%	11.50%
16 weeks	99.29%	30.40%	99.29%	14.11%	98.72%	8.52%

Table 4.12.: Comparison of Personal and Local Blacklists in Good Classified MIT Weeks: *The table lists the percentages of people providing and having the first content when combining a send rate limitation of one content/day, an individual detection recognition of 10% and exchaning blacklist information with a suggestion threshold of 10 and 20 opinions. The week times are relative to day 0 and the percentages to the people having the content with normal spreading.*

classification	time threshold 20	time threshold 10
good	day 17	day 3
average	day 23	day 9
bad	day 57	day 32

Table 4.13.: Suggestion Threshold Barrier: *The table lists the times when a threshold of 20 and 10 opinions is reached in good, average and bad classified MIT weeks.*

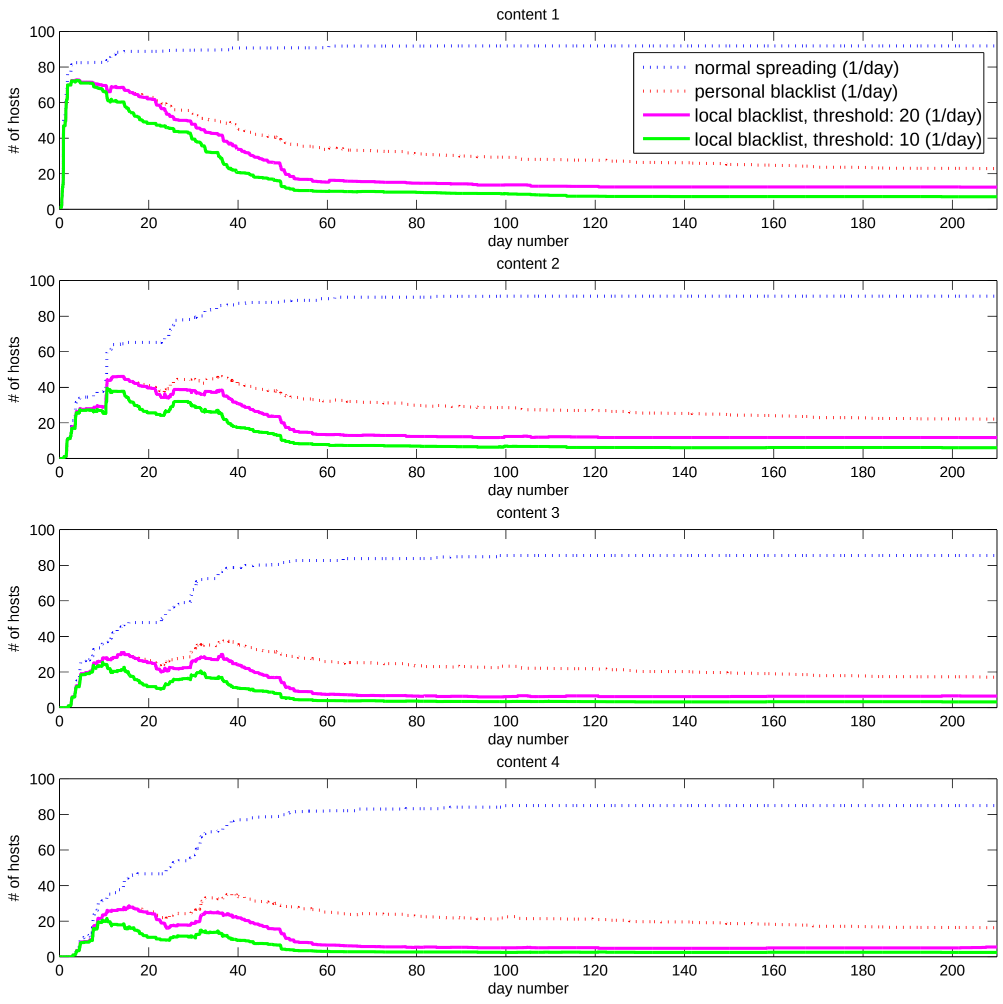


Figure 4.8.: Comparison of Personal and Local Blacklists in Good Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every day. Personal and Local Blacklisting is combined with a send rate limitation of 1 content/day. Every user individually detects spam with a recognition probability of 10%. Additionally, suggestion thresholds of 20 and 10 opinions are used. The graph shows the percentages of people providing the first four contents generated in the first four weeks.*

Since the time scale of dissemination in the MIT sets is quite high due to the few number of connections, we compared local blacklisting with personal blacklisting also in the Haggle data set and marked 8 hours of inactivity as night shifts. The result can be found in Figure 4.9. Similar to the MIT set, we see that personal blacklisting is not efficient at all and exchanging the information about blocked peers could significantly decrease the amount of users providing the content. Whereas with personal blacklists still 62.88% of the people provide the content after 72 hours, this percentage would be decreased to 10.39% and 3.61% with suggestion thresholds of 8 and 4 respectively[23]. When neglegting the night shift, the simulation results with the Haggle set show a performance similar to the generated traces as Appendix C.1.5 shows.

However, decreasing the suggestion threshold increases the susceptibility to liars. As presented in Section 2.4, a mechanism that addresses the lying about ratings is MobiRate. It ensures that a user cannot lie about content that he/she did not receive by using hash lists and selecting witness users. Unfortunately, MobiRate is not suitable in the PodNet environment because of two main reasons. Firstly, we want to rate the authors which are not necessarily the suppliers of the content. Secondly, selecting reliable witness nodes is an unsolved problem. But even if MobiRate would work, it would still be possible to lie about the rating of a content that one has actually received, e.g. by giving a good instead of a bad rating and vice versa. In the next section, we therefore address the problem by weighting the recommendations from community members with higher trust weights as explained in Subsection 3.5.5.

[23] A suggestion threshold of 8 correponds to $\sim$ 20% of the total of nodes and 4 corresponds to $\sim$ 10% which are comparable percentages to the MIT set.

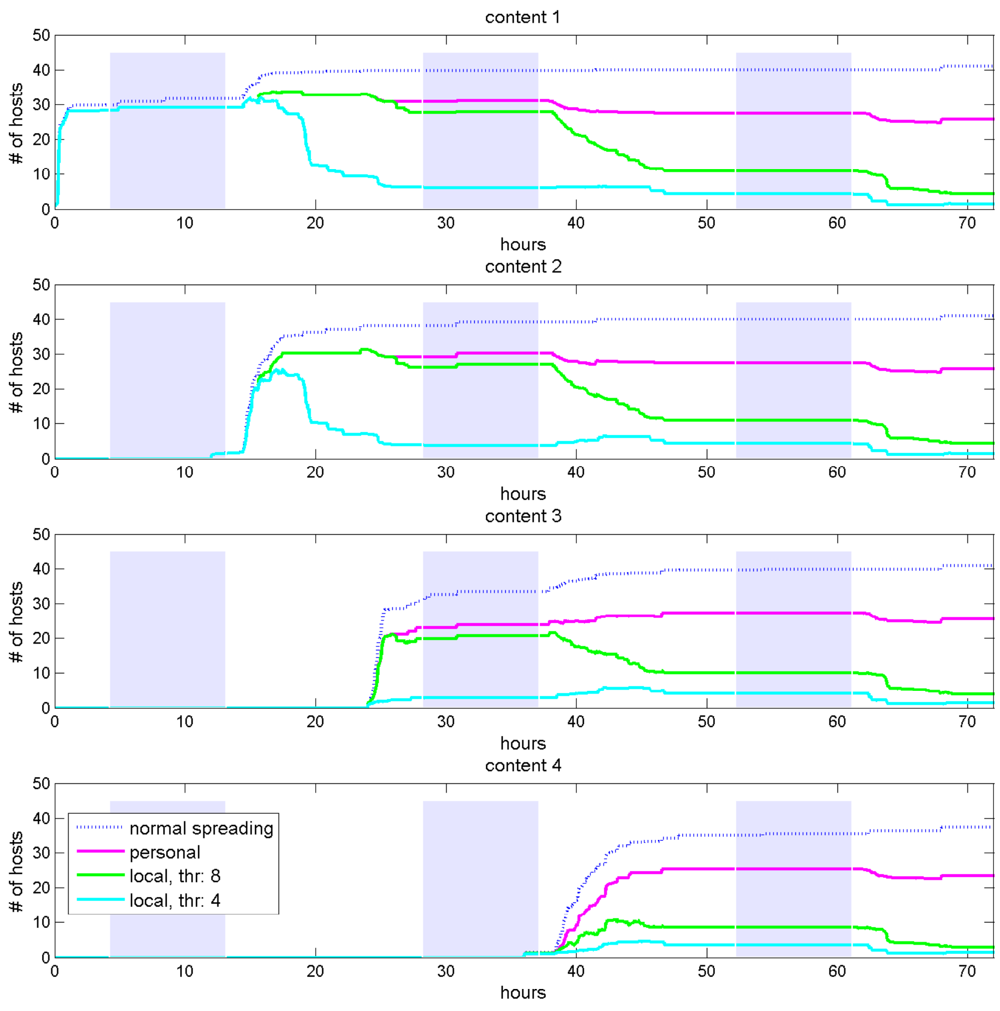


Figure 4.9.: Comparison of the Personal and Local Blacklists in the Haggle Set: *Average spreading performance of one content generated by a spammer every 12 hours. The percentage of people providing content is compared between Personal and Local Blacklisting. Every user individually detects spam with a recognition probability of 10%. Additionally, suggestion thresholds of 8 (∼ 20% of total set) and 4 (∼ 10%) opinions are used. The gray shaded regions show an 8 hour inactivivity period during night. The graph presents the percentages of people providing the first four contents generated in the first 36 hours.*

4.4.6. Influence of Social Weights

In the last section, we observed that decreasing the suggestion threshold leads to a faster spam recognition. As already mentioned above, reducing the threshold value facilitates wrong reputation spreading by attackers. Therefore, we suggested to weight recommendations of community members with a higher weight.

Simulations with Community Detection	
	Unlassified MIT
Parameters	23 spammer seeds *in every round:* 1 spammer 1 content/week p_{rec}: 10%, 20% th_{sugg}: 20, 30, 40 weights: community: 10 familiars: 5 others: 1
Figures	p_{rec} 10: Fig. 4.10 p_{rec} 20: Fig. C.37

Table 4.14.: Overview over all Simulation Parameters with Social Weights.

In this section we will show the results of simulating local blacklisting with community weights which are received by using a modified version of the *Simple* algorithm[24]. The evaluation of the algorithm parameters can be found in Subsection 6.4.1.
When classifying the different weeks into good, bad and average, the underlying social pattern was destroyed by our classification process. Therefore, we simulated the local blacklists with community weighted recommendations as specified in Table 4.14 only in the unclassified MIT data set.

In Figure 4.10 we compare social thresholds[25] of 20, 30 and 40 recommendations to the suggestion threshold th_{sugg} of 20 used previously. As Table 4.15 shows, the maximum peak of people providing content would be decreased significantly even when using a social suggestion threshold of double the fixed value. However, when increasing the social threshold, people which only belong to a small community or none at all, will have more difficulties in blocking the author. As mentioned in Subsection 4.4.3.2, this is not as bad as it seems since the content will not spread widely anymore because most users may have blocked the author already. However, in order not to discriminate against small communities, a tradeoff between threshold security and average community size has to be found.

[24]See Subsection 3.5.4 for more information.
[25]Thresholds which considers the social relationships by community weights. Familiars and community members have a weight of 5 and 10 respectively.

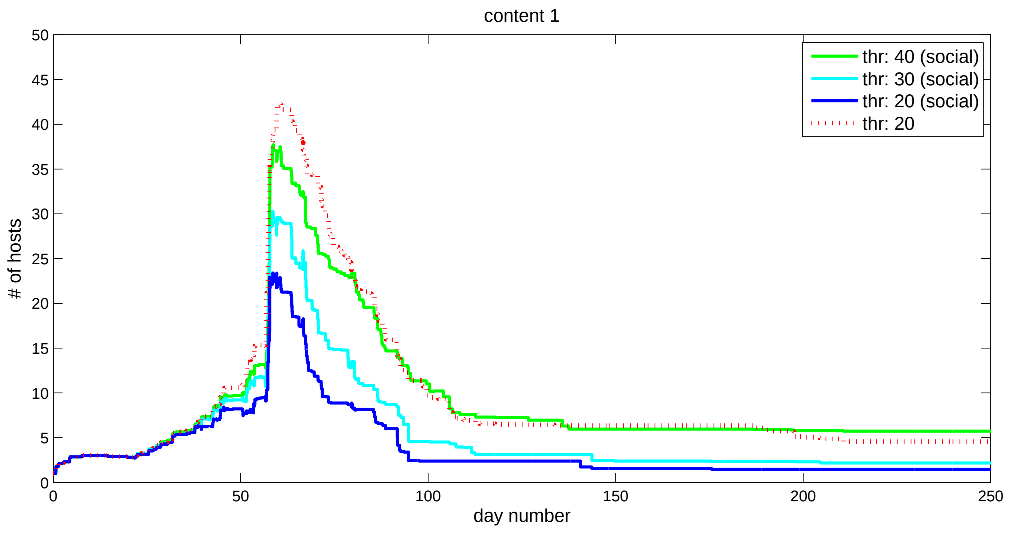


Figure 4.10.: Community Weighted Recommendations: *The percentage of people providing content is compared between a fixed suggestion threshold of 20, and community weighted thresholds of 20, 30 and 40 opinions for a spam recognition of 10%.*

time	E20 (social)	E30 (social)	E40 (social)
day 61 (peak)	50.36%	69.82%	83.73%
day 250	32.38%	47.62%	125.71%

Table 4.15.: Comparison of Social Thresholds: *The effect on people providing content is shown for three social thresholds relative to the fixed threshold of 20 opinions. The spam recognition is set to 10%.*

5. Implementation

The basic PodNet application was created in a previous master thesis by Clemens Wacha and is documented in [3]. In this chapter we highlight the most important changes to the code. Before implementing all the security related features we had to choose and adapt a cryptographic library. In the implementation itself the biggest changes were preformed to the data storage module and the communication process. At the end of this chapter we also suggest improvements to the existing code.

5.1. Testbed

The original implementation was tested on a variety of platforms. However, time only allowed us to test the modified application on Windows CE based systems. The actual PodNet implementation was successfully tested on Windows Mobile 6 running on a HTC Touch and Windows Mobile 2003 running on a HP iPAQ. The details about the devices can be found on Table 5.1.

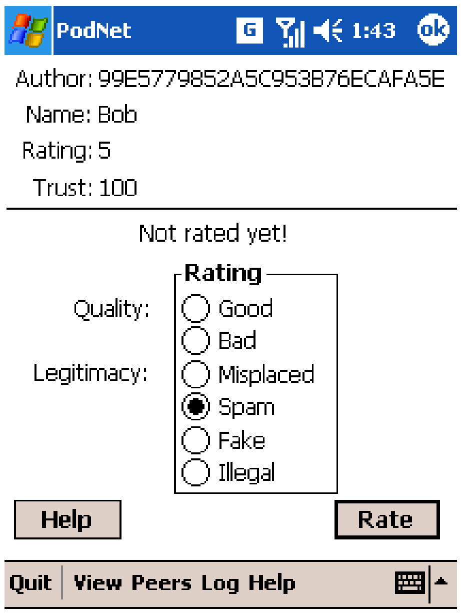
device type	**HP iPAQ hx2410**	**HTC Touch**
processor	Intel(R) PXA270 520MHz	OMAP 850 201MHz
RAM memory	59.55MB	47.93MB
operating system	Windows Mobile 2003	Windows Mobile 6

Table 5.1.: Details about the HP iPAQ and HTC Touch

5.2. Functionality Overview

At first program startup, the device automatically generates credentials for the device owner which basically comprise a public/private key pair and a user prompted name. The name is used as human readable identifier within PodNet but a user will be identified anonymously by the hash of his/her public key. After the first program start, whenever the application is closed and restarted, the credentials, among other saved information, are loaded in order to keep the same user identity.

The basic interface has not changed extremely, the creation of channels and episodes still works the same way as well as the discovery of and the subscription to new channels. There are mainly three new functionalities which are worth mentioning in detail:



Author: 99E5779852A5C953B76ECAFA5E
Name: Bob
Rating: 5
Trust: 100

Not rated yet!

Rating
Quality: Good Bad
Legitimacy: Misplaced [✓ Spam] Fake Illegal

Figure 5.1.: PodNet Rating Interface.

- A *Rating Interface* was added. It allows a user to rate an episode which has the effect described in Section 3.6 and 3.7. Every episode can only be rated once and the rating will be aggregated to its author in the channel. The rating interface can be seen in Figure 5.1.

- Another enhancement is the *Channel Management*. The owner of a channel can add and remove moderators as well as include members or put users on the channel blacklist depending on the channel class. A moderator has the same rights to manage the members and the channel blacklist as the owner.

- Further, the *Secure Pairing*[1] can be initiated from the peers dialog in order to establish a friendship with a neighboring peer.

All other new interfaces only provide additional information like trust values of all users seen so far as well as rating values and episode list from an author in a particular channel.

5.3. Cryptographic Library

As explained in Chapter 3 we require both symmetric and asymmetric cryptography as well as hashing for the security extensions. Asymmetric cryptography is needed for authentication in the challenge response mechanism as well as to sign episodes and the different lists. Symmetric cryptography on the other hand is used in secure pairing to encrypt a challenge with a given password as well as to encrypt files in closed channels.

[1]See *Secure Pairing* in Subsection 3.3.2.

We decided to use the Crypto++ library[2] version 5.5.2 which provides the most common cryptographic algorithms. Its wildly usage in practice, its FIPS compliance and platform independence made it our first choice. The library has been ported to Windows Mobile 2005[3] and we modified it so it would run on Windows Mobile Pocket PC 2003. We considered the *Advanced Encryption Standard (AES)* for symmetric encryption, the *Secure Hash Algorithm (SHA)* for hashing and compared the performance of RSA [35] and *Elliptic Curve Cryptography (ECC)* [36] for the asymmetric cryptographic operations. The evaluation of the cryptographic library can be found in Section 6.1.

5.4. Communication Process

In this section the changes performed to the communication process are discussed. In particular, there is the service discovery including the peer selection process which leads to the actual content synchronization in the transfer protocol as well as the manually invoked secure pairing with the purpose of forming social connections.

5.4.1. Service Discovery

In order for PodNet to work in an opportunistic network service discovery packets have to be broadcasted periodically. As described in [3] these packets are sent every 2 seconds and contain every peer's ID[4], a service mask and 3 timestamps.
Besides the existing *'new content'*-timestamp, another one for the community lists as well as one for the global blacklists was added. They are both used to indicate whether an update of those lists is needed. In order to keep the discovery packet small we abandoned the 19 byte string encoding of the timestamp[5] used before and only used 4 bytes to represent the seconds since 1970.

The service mask is used to advertise available services by the corresponding peer. It is currently only used to announce the podcast service itself and to state whether the peer holds a CA signed certificate. This way, a neighbor can request a credentials update from a peer that has recently obtained an additional CA signature. Additional services that might be useful are providing Internet connection, a registration service for a CA or even forwarding service in case routing is allowed.

Every peer collects all the available discovery packets and comes up with a peer list. Whenever more than one peer is available on the peer list one of them has to be selected to start the content synchronization process. In the previous implementation this selection procedure

[2]http://www.cryptopp.com/
[3]http://www.ugosweb.com/cryptopp.aspx
[4]The random string used in the previous implementation is replaced by the hash of the public key of the peer's credentials.
[5]See Section 5.7.

was random with equally distributed probabilities for each peer. However, it is a good idea to consider a peer's trust value[6] when selecting a neighbor. It is obviously preferable to synchronize content with previously known and/or even trusted peers since we know already something about their behavior. On the other hand, if a peer is blocked[7] for some reason, we don't want to connect to it at all.

Algorithm 5 Peer Selection

```
$n$ = number of active peers
**for all** peers $p_i$ such that $1 \leq i \leq n$ **do**
  store $p_i^{\prime}s$ trust value $t_i$
**end for**
sum $s_0 = 0$
**for** $i = 1$ to $n$ **do**
  $s_i = s_{i-1} + t_i$
**end for**
generate a random equally distributed number r in the range $[0, s_n]$
**for** $i = 1$ to $n$ **do**
  **if** $r < s_i$ **then**
    select $p_i$
  **end if**
**end for**
```

In order to select a peer for connection, Algorithm 5 is used. It chooses a peer p_i with a probability according to his/her trust value t_i. The three for-loops have a complexity of $O(n)$ each, where n is the number of active peers, and all other statements are done in constant time. Thus the resulting complexity of the algorithm is $O(n)$.
Although we only considered the trust value, other factors, like providing special services[8] or having a good author rating[9] might be taken into consideration when selecting a peer.

5.4.2. Content Synchronization

Whenever content is exchanged between users a transfer protocol is executed. How much and what content is exchanged is defined by the download policy. In our implementation the protocol had to be modified slightly as well as some policies had to be added. The changes are described in this subsection.

5.4.2.1. Transfer Protocol

The former basic mechanism to synchronize content can be separated into eight stages [3]. In order to guarantee a proper authentication and authorization we had to modify several of them

[6]See Section 3.5.
[7]Blocked users have trust value 0.
[8]See beginning of this subsection.
[9]In case the peer is an author.

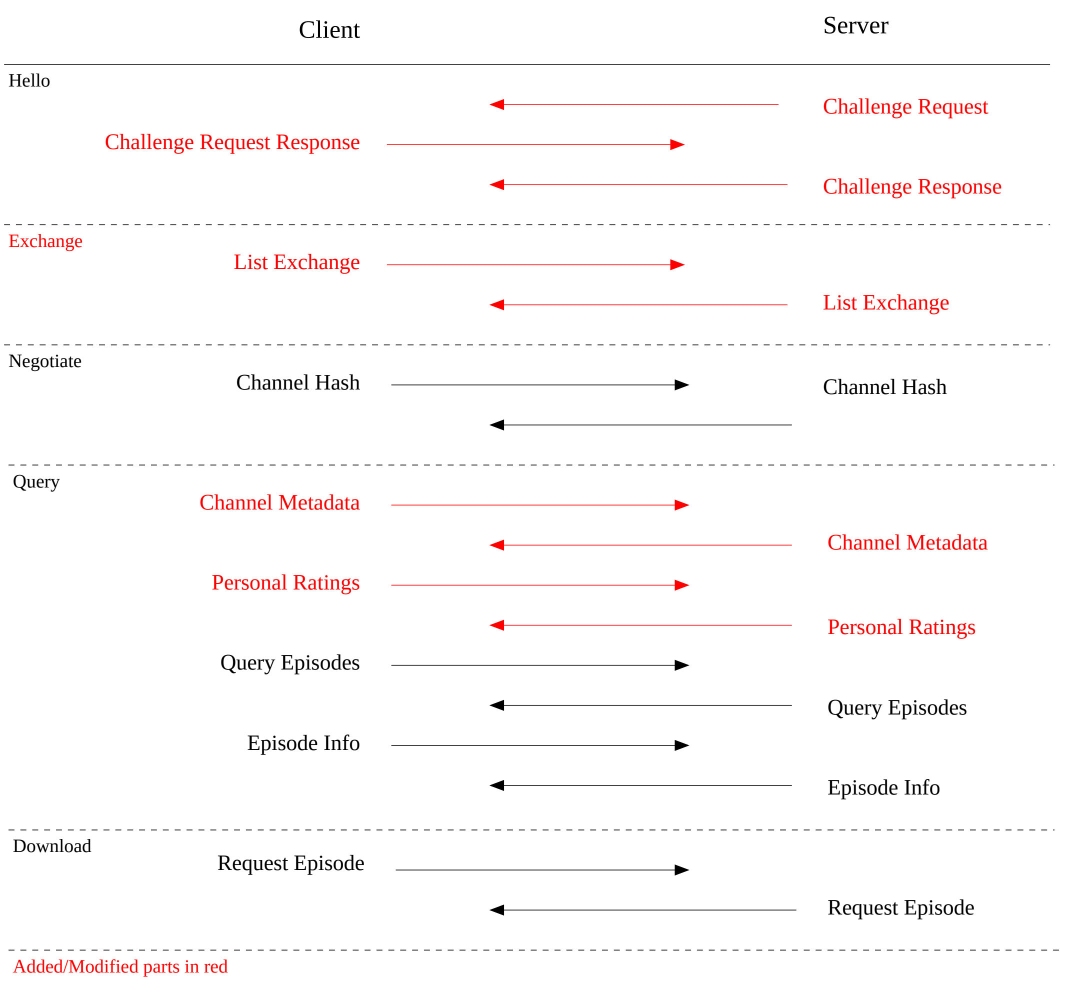


Figure 5.2.: Communication Protocol

as well as introduce the 'EXCHANGE' stage as explained below[10]. The important changes to the transfer protocol are shown in Figure 5.2 where the red colored parts point at differences compared to the basic implementation in [3].

HELLO: In case the peers do not have each other's credentials, they will be exchanged and challenged in a challenge request response scheme. The existing mechanism which exchanges only a hello packet in both ways is still present but only used if the two communicating peers have each others credentials already.

EXCHANGE: This stage only exists if either the friends list, the familiar list, the community list or the global blacklist of the corresponding peer is outdated. If this is the case the required

[10]Visualizations pointing at the implementation changes can be found in Appendix D.1.

list is exchanged and updated. In order to know whether the lists need an update, timestamps are sent in the discovery packets.

QUERY: The ‘QUERY’ stage is already used to request and exchange the episode list of the desired channels. Additionally, the channel meta data as well as the users rating and blacklist recommendations are exchanged for the requested channels. Note that the additional information is only exchanged if the downloading peer does not possess the newest version of it. In order to avoid sending this information multiple times, information timestamps are exchanged. In case episodes are desired for download after this stage, the supplier, if not already done during the ‘HELLO’ stage, has to be authenticated before entering the ‘DOWNLOAD’ stage.

The most significant change of the transfer protocol is the introduction of an authentication by a challenge request response mechanism.
If the authentication is performed during the ‘HELLO’ stage, it ensures that the peer is the actual owner of the credentials he/she presents. In case it is done in the ‘QUERY’ stage before downloading content the reason is a different one. The idea is to make sure we know from whom we get content. For example, if the hash of the received data chunk results to be corrupted, the specific peer could be punished or blocked[11]. This can only be done if they are properly authenticated before data transfer.
Note that in the current implementation the hash check is not performed over the different chunks but over the whole file. Additionally, no action is taken when the hash check fails since the hash check occasionally fails for unexplainable reasons although the received file seems to be correct.

5.4.2.2. Download Policy

The previous implementation introduced a download policy per channel. It allowed the user to choose between the policies *all*, *manual* and *newest*. As the name already implies, download policy *all* decides that all the content in a channel should be downloaded whereas under the *manual* download policy, only the meta data of an episode is downloaded and the user decides whether to download the actual data or not. Download policy *newest*, although present in the GUI, was not implemented at all, thus in practice it would behave as in the *manual* policy. We did not modify the functionality of those existing policies, although the *newest* policy where only recently published content would be downloaded would make a lot of sense, in particular when one could determine how new the desired content should be.

In order to make use of an author’s reputation, a new default policy[12] is introduced, namely *automatic*. Under this policy we check that an author is not blocked for any reason and rate limit is not reached[13] before downloading the episode.

[11]Optionally, one could think of rewarding peers one gets a lot of valid content from.
[12]Replacing former default policy *all*.
[13]See Section 3.7.

Additionally, the policy *none* was added, which does not download any content or episode meta data at all but only collects reputation information from authors in the channel. This way a user may explore the quality of a channel without downloading any content.

5.4.3. Secure Pairing

The secure pairing is performed in order to prove friendship by signing each other's credentials in a secure way. This is achieved through a basic three way challenge response request protocol based on a commonly known password.
The user initiating the pairing procedure encrypts his/her user ID with the password as proof of the knowledge of the secret key and sends his/her credentials together with the encrypted string to the other user. Upon reception the other user verifies the decrypted user ID. If the verification is successful, the received credentials are signed and the signature is sent back together with his/her own credentials and encrypted user ID.
In case the initiating user receives the correct user ID, the signature of the newly acquired friend's credentials is returned and the received friend signature is added on the friends list. From that moment on, both users can proof their friendship by presenting the friend's signature.

5.5. Data Storage Module

The *'Datastore'* module of the PodNet implementation increased dramatically in size and complexity since it no longer had to hold only the channels and episodes but also information about users and the community. Additionally, the new data structures are much more dynamic and time dependent, thus periodically aging the *'Datastore'* becomes necessary. Besides the core functionality, this module also collects statistical data which could be useful for future applications.

5.5.1. Structure

The *'Datastore'* was used to contain only channels and episodes, i.e. the actual content that is shared and distributed in PodNet. However, in order to introduce a notion of a user and to apply the proposed security measures, the existing structures had not only to be enhanced, but new structures to save additional data, such as user specific information, had to be defined. The complete structure of the new *'Datastore'* module can be seen in Figure 5.3. Each of the submodules are explained in the following.

Credentials The *'Credentials'* module assures the basic authentication[14]. For simplicity reasons the same module was used for both users and channels. Apart from the required information like name, ID, public key, the credentials provide an interface for the basic verification and encryption functions as well as the signing and decryption operations in case the private key is available.

[14]See Section 3.3.

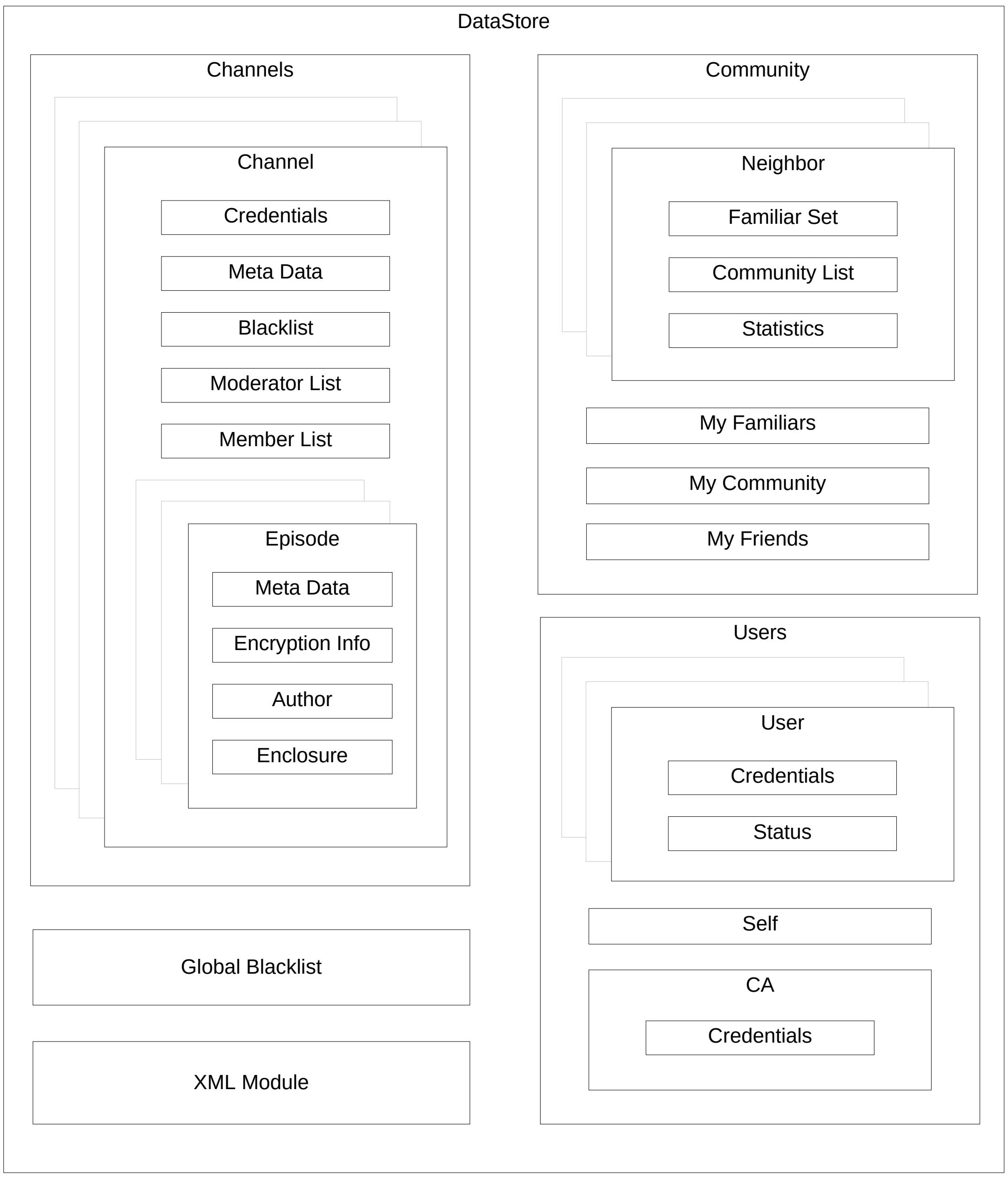


Figure 5.3.: The Datastore Module.

Users This module manages all the *'User'* structures which store information about every entity one comes into contact with. The least a *'User'* structure requires is a user ID but in case more information is known, also the user's *'Credentials'* may be stored in it. In case the user is a neighbor, it contains the *'Neighbor'* module described below. For every channel in which the user publishes content, an *'Author'* module is added to the *'User'*.

Neighbor In the *'Neighbor'* module all statistical contact data[15] for encountered users including all his/her social data like friends and community members are saved. This information is used by the community detection algorithm[16] as well as to determine the trust in the encountered user.

Author An *'Author'* module exists per *'User'* and *'Channel'* in case the user has created episodes in the specific channel. Each episode is linked to the *'Author'*. The module contains all the reputation data collected about the user such as personal and local rating and blacklist.

Channels The *'Channel'* structure already existed in the original implementation with the purpose of managing all the episodes, but is enhanced and additionally contains a *'Credentials'* module in order to assure the authority in a channel. The channel meta data were slightly modified and the authorization lists[17] were added as well in order to manage the users of a channel.

Episodes The *'Episode'* structure which also already existed in the original implementation is used to store the actual content. The episode meta data were only slightly modified but its ID is cryptographically bound to the channel it is published in and the author it is created by. Additionally, in case the episode is included in a closed channel, the episodes encryption info is stored in this structure as well.

Community This module is in charge of the community detection and friend circle construction algorithms. It keeps track of the active peers in order to allow recording of connection statistics by the *'Neighbor'* module. It updates and ages the community and familiar sets periodically and builds up the friend circle whenever friends lists are received from neighbors.

Global Blacklist The global blacklist is applied against all authors in all channels and therefore, this Module is at the lowest level of the *'Datastore'*. It contains the list of blacklisted user IDs and should be timestamped and signed by a CA.

XML Module The sole purpose of this module is to preform all the XML encoding and decoding for all the different modules in the *'Datastore'*. The reason this functionality was outsourced is the drastically increased amount of functions compared to the original implementation.

[15]See Subsection 5.5.3.
[16]See Subsection 3.5.4.
[17]See Subsection 3.4.3.

5.5.2. Aging

In order for the proposed security feature to work properly as well as to keep the size of the *'Datastore'* under control, an aging mechanism had to be introduced. Mostly user related information is aged. The aging is performed every hour although only neighbor information has to be aged that frequently since it is required for the community detection as described in Subsection 3.5.4. Author related information usually ages on a daily basis as all the timeout values for blacklists, recommendations and ratings are specified in multiples of days as described in Section 3.6.4. Each user entry itself, specially if the user is not an author and does not contain any additional neighbor information, should age and be deleted upon a timeout to avoid saving peers one will never hear from again.
The aging in the *'Datastore'* also has to make sure that rate limitations, which ensure that only a certain amount of content is downloaded per author or channel[18], are reset every day.

5.5.3. Statistics Collection

Whenever a neighbor peer is discovered we start to save statistical data about the encounter. Currently, the only data that is used is the total contact time. This information is required for community detection[19]. In fact much more detailed statistics are actually saved. For every encounter with every node the start end times are saved. Additionally, the number of encounters as well as the time the peer was seen for the first and for the last time is recorded. If a future community detection mechanism or any other algorithm should need such information, it can be found in the *'Neighbor'* module. Additionally, it could be useful to collect other information as well. Such information may comprise the frequency of connections compared to the actual data transfers with a peer. This information, as well as detailed data upload and download statistics could be used to influence the peer selection process[20] and increase fairness in the content distribution. All the statistical information is summarized in Table 5.2.

Value	**Description**
`time_seen_total`	Total time the peer was in proximity
`time_seen_aged`	Total time reduced by aging (for community detection)
`count_seen_total`	Total amount of times the peer was seen
`connection_times`	Start/end time pairs of each time peer was in proximity
`first_seen`	First time the peer was encountered
`last_seen`	Last Time the peer was seen
`times_connected`	Amount of times synchronized with peer (unimplemented)
`times_data_exchanged`	Amount of times data exchanged with peer (unimplemented)
`data_received`	Amount of data received from peer (unimplemented)
`data_sent`	Amount of data sent to the peer (unimplemented)

Table 5.2.: Collected Statistics.

[18]See rate limitation for unknown authors in Subsection 3.7.1 under *Rate Limiting*.
[19]See Subsection 3.5.4.
[20]See Subsection 5.4.1.

5.6. Discovery Channel

In Clemens Wacha's implementation [3] the discovery channel is implemented as a hard-coded regular channel which all users are subscribed to. Basic channel information of known channels is included as episodes. When two peers meet, they always exchange all the episodes in the discovery channel independent of whether the channels are locally available or not. A peer can then subscribe to one of the channels in the discovery channel and will download the content the next time he/she sees the channel.
This transparent handling of the channel discovery by abusing the channel/episode infrastructure is no longer useful in the current implementation since the infrastructure became much more complex. Considering who the owner of this channel should be and why this channel needs credentials results in the conclusion that the channel discovery mechanism has to be revised. Due to the limited time frame we did not redesign the discovery channel.

It is also obvious that the current method does not scale since every user's discovery channel will tend to include all possible channels even though they have never been seen. As a consequence, the size of the discovery channel will increase dramatically which handicaps the channel handling. Finding interesting channels and subscribing to them becomes increasingly difficult the more channels are available.

One could tackle the problem in many ways. It would help if responses to channel discovery request only comprise the channels which are locally available. Additionally, the received channels in the discovery channel could age and be removed if they are not subscribed for a certain time. Optionally, if a channel is seen regularly, the aging timeout could be reset so that the most common channels would not be removed. To reduce sending overhead, one could introduce maximum accepted discovery channel sizes which would limit a user to completely flood another user with channels. Another option would be to have a channel rating or at least some kind of availability information. An accurate and relevant channel rating would be tricky to achieve but the availability can easily be detected by counting the number of distinct users offering the channel. On the other hand, a user could be interested in a channel which has a low availability. For this reason it could also be interesting to try to bring content with low availability to the interested user instead of removing it from the Discovery Channel.

Another, more rudimentary option, as specified in the concept in [2], would be to limit the spreading of channels in the discovery channel by only exchanging them on demand and not during each data transfer as in the current implementation. A user that is interested in new channels could then request the available channels from the discovery channel of other users.
Nevertheless, this scalability issue is out of the scope of this thesis and was thus not considered.

5.7. Potential Improvements

In this section, we present some aspect of the current PodNet implementation which in our opinion should be improved. It is organized into two parts. The first part contains some structure related improvements whereas the second part explains some optimizations for the communication process.

5.7.1. Application Structure

The current breakdown of the functionality into the different modules does make sense but in the actual implementation modularity was sacrificed by disregarding encapsulation completely. Hardly any attributes are private thus the modules lack a proper interface. This is important because anybody that tries to make a small change to a simple unimportant data member of a specific module has to change half the application. For example if one changes a field in the meta data of an episode one would not only break code in the *'Datastore'* module but also the transfer infrastructure and the graphical userface.
This problem mostly concerns the data storage module. It even got worse with the added functionality, specially because a lot of the new structures are linked between each other. In particular, the *'User'* module is not solved optimally. Designing a proper *'Datastore'* is not an easy but necessary task since it is used intensively by the rest of the application. Another issue that should be mentioned is the significant amount of dynamic memory used and shared among the submodules inside the *'Datastore'* module. This memory, as well as the mutexes used to ensure that only one thread is accessing the data, should be handled by smart pointers in order to prevent memory leaks and deadlocks.

In order to access every module from everywhere the global 'app' pointer was introduced in the original implementation. This destroys modularity completely. An easy fix would be to introduce a module communication bus or a central communication module holding all the message queues, subscription services of the different modules. The global rc_commands which use the the 'app' pointer in order to call member functions of the different modules, should be implemented as functions in the scope of the class they actually belong to.

There are a few other issues concerning the application structure that are worth mentioning. The router module should be a separate thread. Firstly because it is a communicating part of the application, secondly because it crashes sometimes. If that happens, it would at least not block the rest of the application, one could kill it and restart it without the user noticing.
Another issue is the extensive use of inheritance in this implementation. There is no reason for example why the 'SyncManager', the 'Router' and the 'SimpleRPC' module have to be a socket. If they would just have a socket, it would make more sense and the code would be more intuitive as well.
The cryptographic wrapper could be improved as well. The PodNet implementation does not

contain any custom made exceptions. Error handling is done by traditional conditional statements, return values and log messages. The Crypto++ library however throws a lot of exceptions. It does not only do so when serious errors occur but also when the library is used properly e.g. when trying to decrypt a corrupted file. Unfortunately this was discovered too late so proper handling was not included in the wrapper when it was written. For this reason exceptions are caught manually when using the wrapper. Clients of the wrapper should not have to worry about such details, thus such matters should be dealt with internally.
Apart from the issue with the exceptions the implemented wrapper lacks the ability to change algorithms and key size at runtime. Finding an alternative to the preprocessor directives which switch the algorithm, would solve this issue.

5.7.2. Communication Process

The communication process has several drawbacks. Firstly, it is a semi duplex procedure. This should be changed by making the sending and receiving part totally independent from each other which can be done by introducing two separate message queues and would make the whole process more flexible. This way, one would not be forced to get content or at least check whether there is available content from users that establish a connection with us, which also makes sense from a security point of view. Besides, it increases fairness since user could then download content from each other simultaneously.

Another optimization would be avoiding the use of XML to send any data. Not only because the tags require unnecessary space, but also because a lot of random strings like IDs and signatures are exchanged and they require a lot of escaping. Currently all random strings are hex encoded which takes the double amount of space. All of this overhead could be reduced by introducing a better format.
An additional small improvement would be to save the timestams as 4 byte integers instead of strings in the format ‘yyyy-mm-dd-hh:mm:ss’ consisting of 19 bytes and requiring string parsing.

6. Evaluation

6.1. Cryptographic Functions

We evaluated the overhead in size and processing time on both a HP iPAQ and a HTC Touch. The detailed specifications can be found in Table 5.1.
In Figure 6.1, we compare the times needed on both the iPAQ and HTC to sign and verify a string of length 190 bytes using a key size of 2048 Bit for RSA and 224 Bit for ECC[1]. In general, all test on cryptographic functions performed better on the iPAQ because of stronger processing power.
When using RSA, the processing time of private key and public key operations varied a lot whereas in ECC they stayed approximately at the same level. The signing of a string with RSA took more than three times longer than with ECC but on the other hand, verifying was approximately 15 times faster with RSA. On average, the RSA cryptography takes more time. However, we assume that people will generate new content rather seldom in the PodNet environment compared to the amount of content received which has to be checked for authenticity. In Figure 6.1 the time it takes to perform different cryptographic operations on the mobile devices is presented. The group of bars on the right side show that signing and five times verifying of a string takes more than twice the time with ECC than with RSA. Thus, the figure indicates that RSA cryptography would be much more efficient in such an environment which motivated us to use RSA in the current implementation.

However, increasing the key length size in RSA has a much higher impact on processing and storage overhead than in ECC since the key size increases exponentially compared to ECC as the specifications from [37] in Table 6.1 show.

ECC	RSA
160	1024
224	2048
384	7680

Table 6.1.: Public Key sizes of ECC and RSA in bits corresponding to equal key strength.

In the current implementation, we reduced the RSA key size from 2048 to 1024 bit[2] to limit the

[1]RSA 2048 and ECC 224 are considered equally secure despite the key size difference.
[2]We are aware of the fact that a key size of 1024 bit is too short to be considered secure since [38] shows that with substantial effort and costly equipment, it can be broken in less than a year. We reduced it for this prototype due to speed improvements.

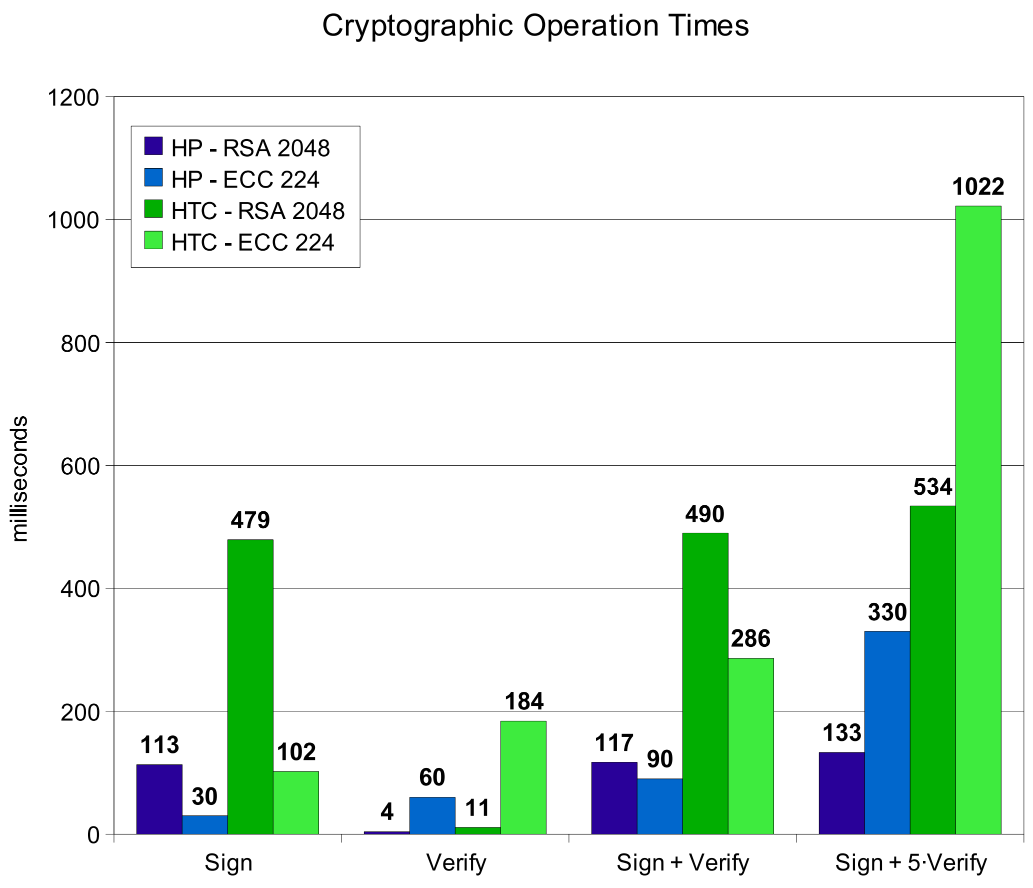


Figure 6.1.: Performance Evaluation on HP iPAQ and HTC Touch.

sending overhead to use RSA key sizes[3]. When increasing the key size to a value larger than 2048 in future implementations, a reconsideration of ECC may be needed because of it's smaller key and signature size as well as faster computation times as [39] shows.
Since signatures and identities[4] are frequently exchanged, we chose to use the SHA1 hashing algorithm in order to reduce the sending overhead. It produces digest lengths of 20Bytes which are much shorter than stronger SHA versions but at the expense of a weaker security. Symmetric cryptography is performed with AES 128 which could easily be extended to 256Bit but guarantees enough security at the moment.

6.2. Communication Overhead

It is difficult to compare the communication of the Secure PodNet implementation with the original implementation because the overhead depends on the exchanged list sizes which vary in size with the number of entries. Therefore, we excluded the list exchange from our overhead comparison but address them in Subsection 6.2.1. We evaluated the overhead of the basic data transfer mechanisms by observing the network traffic with Wireshark[5]. The detailed results can be found in Appendix E.2. We will divide overhead observations for a file with size $\sim 26KB$ into the three stages, i.e. *Hello*, *Query* and *Download* procedure which can be found in Table 6.2. The remaining stages, i.e. *Negotiate* and *Done* comprise no overhead.

	Original Implementation	**Current Implementation**	**Overhead**
Hello	52 Bytes (2 msg)	*without Auth:* 56 Bytes (2 msg)	7.69%
		with Mutual Auth: 2026 Bytes (3 msg)	379.96%
Query	*No Channel:* 122 Bytes (4 msg)	*No Channel:* 188 Bytes (4 msg)	54.09%
	1 Channel: 1121 Bytes (8 msg)	*1 Channel (with meta):* 3075 Bytes (10 msg)	174.30%
		1 Channel (normal): 1502 Bytes (8 msg)	33.99%
Download	*1 Episode:* 27456 Bytes (57 msg)	*1 Episode (with Auth):* 27608 Bytes (41 msg)	0.55%
		1 Episode (without Auth): 27466 Bytes (39 msg)	0.04%

Table 6.2.: Communication Overhead of TCP Data.

When neglecting the exchange of any lists, a user that connects to an unknown device and downloads a file of size $\sim 26KB$ from an unknown channel would experience a communication overhead of 13.27%. However, in normal operation, when both user know each other and the

[3]Compare with Table E.8.
[4]Identity: hash of public key, refer to Section 3.3.1.
[5]http://www.wireshark.org/

user would download from a channel he/she already holds the current meta data, the overhead would be reduced to only 1.44%. The more content a user will download from the same channel, the smaller the overhead since the sending of channel meta data and the authentication will only be performed once per connection.

6.2.1. List Overhead

All the exchanged lists are sent in separate packets[6]. In the original implementation, the receiver of the transfer client holds partial packets in its receiving buffer until the packet is complete and forwards only completed packets to the Message handler. Due to the fact that the exchanged lists may have varying sizes, we increased the buffer to hold a maximum of 20 Flex packets[7]. In this chapter, we will analyze the entry sizes of the different lists.

Familiar, Community, Blacklist: The list format for the familiar and community set, as well as for the blacklist are all identical. The layout is shown in Table 6.3. The timestamp and signature are generated at creation of the list. Every list entry comprises a 20 Byte peer ID which leads to a maximum of 65 entries per exchanged packet. The maximum list size that would fit in the receiver buffer would contain 1448 different entries. However, because of the aging algorithm in the community detection algorithm[8], the list size should not become that large but may be exchanged quite regularly.

4 Byte	128 Byte	20 Byte	...
Timestamp	Signature	Entry (Peer_ID)	...

Table 6.3.: List Format for Familiar and Community Set as well as Blacklist.

Friends List: The overhead of friends lists is much larger since every list entry additionally contains a 128 Byte signature of the friend. The format is as in Table 6.4. Due to the large size of the friend signatures, only 9 friends can be included in one single packet. However, the maximum buffer size would still hold 196 friend entries at maximum.

4 Byte	20 Byte	128 Byte	...
Timestamp	Peer_ID	Signature	...

Table 6.4.: List Format for Friends List.

Rating List: The format of the rating list is shown in Table 6.5. Compared to the blacklist above, an additional rating value of 1 Byte per entry is added. Therefore, 62 entries can be included in a single packet which corresponds to ratings for 62 different authors per channel. At maximum, 1379 entries can be exchanged.

[6]Except the Moderator list which is exchanged together with the Channel Meta data.
[7]See [3].
[8]See Subsection 6.4.2.

4 Byte	128 Byte	20 Byte	1 Byte	...
Timestamp	Signature	Peer_ID	Rating	...

Table 6.5.: List Format for Rating List.

Global Blacklist: Additionally to the blacklist above, the global blacklist comprises the ID of the central authority (CA). The format in Table 6.6 allows to send a maximum of 64 entries in a single packet and a maximum 1447 entries would fit in the receiving buffer. However, the number of global list entries may be potentially high since the global blacklist is considered in all channels.

4 Byte	20 Byte	128 Byte	20 Byte	...
Timestamp	CA ID	CA Signature	Peer_ID	...

Table 6.6.: List Format for Global Blacklist.

Moderator List: The moderator list is part of the channel meta data, which is stored in XML, and thus shares the timestamp and signature with the other meta data. Every moderator list entry comprises the entire moderator certificate which has a size of $\sim$ 900 Byte. In order to save resources, the channel meta data is therefore only exchanged if the receiver would need it.

Channel Blacklist, Member List: Both lists are XML encoded. Every entry contains all the fields in Table 6.7. Both IDs and the signature are hex encoded which doubles their size and results in an entry size of 341 Bytes. A single data packet would thus only include 4 entries and a maximum of 85 list entries can be exchanged at once.

20 Byte	4 Byte	20 Byte	1 Byte	128 Byte	...
Peer ID	Timestamp	Mod Id	Rights	Mod Signature ...	

Table 6.7.: List Format for Channel Blacklist and Member List.

From all these considerations, it becomes evident that the list encodings needs to be improved since the hex encoding of all peer IDs and signatures in XML doubles the effective entry size and thus increases the exchanged overhead significantly.

6.3. Computational Overhead

In order to evaluate the computational overhead of the current implementation, we measured the total synchronization time as well as the time needed for hashing and decrypting a file in the different channel types. The evaluation was performed in the original[9], open and closed channel by sending either one of the following:

1. No File: only episode meta data exchange (without channel meta data)

[9]By original we mean channels from the original implementation documented in [3].

2. Small File: 52.79 KB in size
3. Large File: 5.99 MB in size

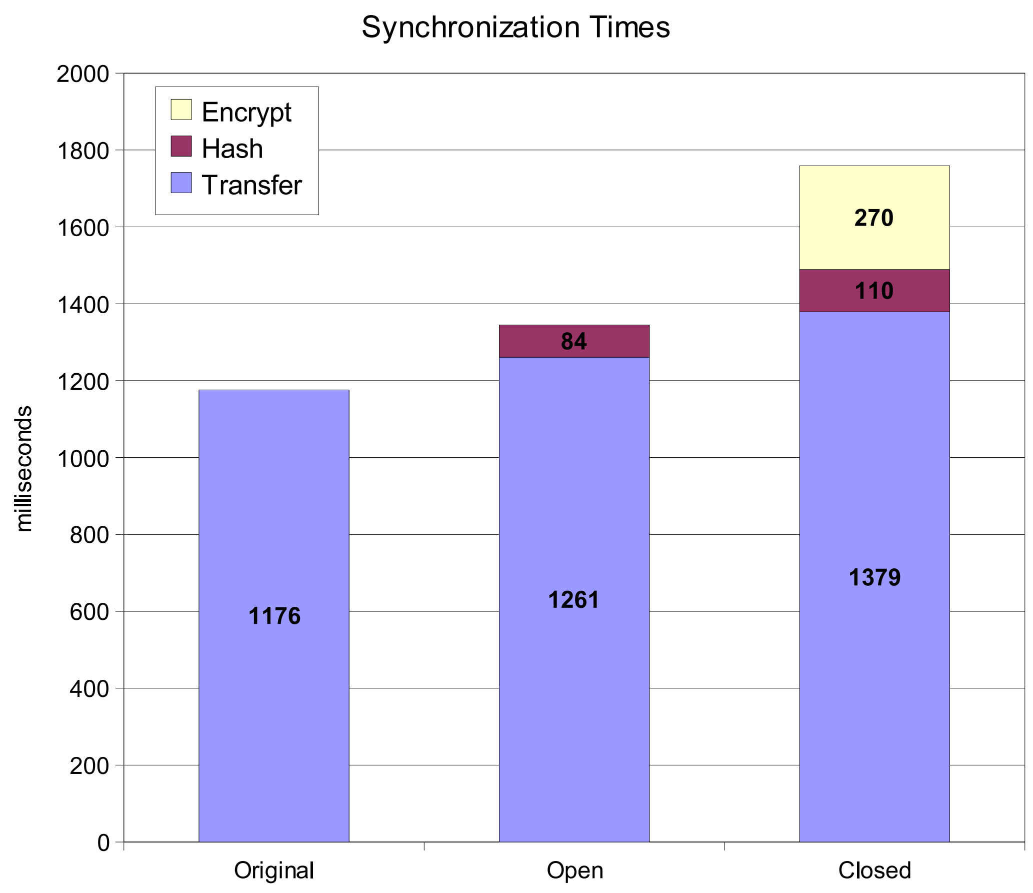


Figure 6.2.: Computational Overhead Sending a Small File.

The measured times varied ∼ 10% although we calculated the average over 100 synchronizations[10]. This was probably due to temperature fluctuation since the devices bacame quite hot and could not be cooled down properly. The surrounding wireless traffic seemed constant at the times meassured so this was probably not the reason. The processing times for the small file are visualized in Figure 6.2. The basic transfer times increases with increased channel complexity because more channel meta data has to be send, i.e. hashes and encryption keys. However, the overhead decreases with increasing file size as Table 6.8 shows.

At first view, this was expected because all processing overhead is sent at the meta data which is independent of the size of the data. However, we observed[11] when subtracting the increased

[10]The detailed data can be found in Table E.18.
[11]See Figure E.18.

Percentage Overhead		
	Open Channel	Closed Channel
No File	1.66%	1.66%
Small File	14.37%	49.57%
Large File	8.26%	50.36%

Table 6.8.: Percentages of computational overhead when exchanging files of different sizes.

hash processing time, the basic transfer processing will be significantly reduced compared to the original implementation. The reason may be the fact that in the original implementation, the maximum packet size exceeded the maximum segment size as explained in Subsection E.2.4 which resulted in splitting the exceeding packet into a large and a small packet.
In contrast to the open channel, the data transfer in a closed channel does not increase with file size because of a large processing overhead for file encryption. We cannot explain this huge overhead since in Table E.7, the processing time for a much larger file using even larger encryption keys was substantially lower. The authentication overhead for the challenge response mechanism varied always between 60 and 130ms for all data transfers and is thus almost neglectable compared to costly cryptographic functions on files such as hashing and encrypting.

6.4. Improved Community Detection

In this section the modification of the community detection algorithm is evaluated and compared to the its original counterpart [24]. In a second part the aging algorithm is analyzed and as a final note some security issues concerning these algorithms are highlighted.

6.4.1. Modified Simple

As mentioned in Subsection 3.5.4 we based the community detection on the *Simple* algorithm. In order to verify the desired behavior we analyzed the algorithm based on the MIT reality traces[12] as well as with a simple real world scenario using the iPAQ devices. We also applied the algorithm on the Haggle traces[12] but since we lacked the information of the social connections no proper analysis could be preformed.

The average community size the *Simple* algorithm detects using different familiar set thresholds in the MIT traces is quite constant as the blue line in Figure 6.3 shows. Nevertheless the size of the individual communities can vary a lot as the yellow and the red line, corresponding to node 39 and 94, shows. The reason can be found in the criterion[13] *Simple* uses to add a familiar node to the community.

In order to visualize the problems with this criterion we constructed the following scenario with

[12]See Section 4.2.
[13]See Equation 3.1.

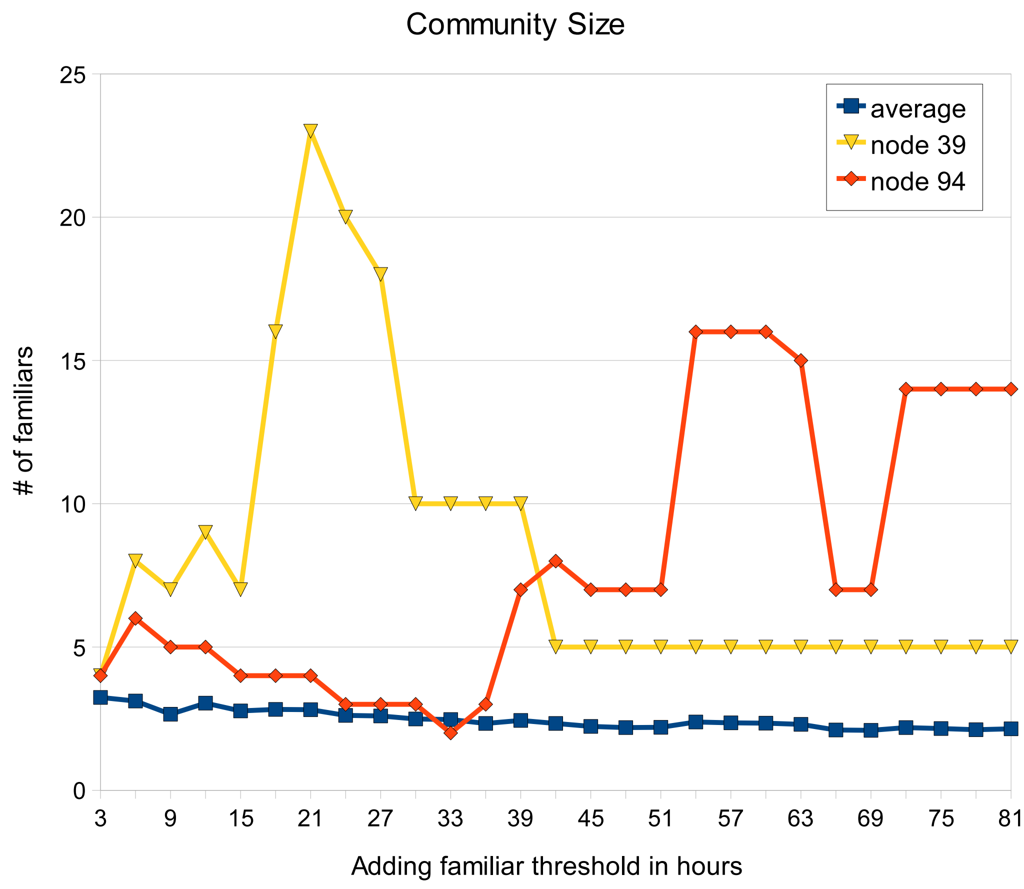


Figure 6.3.: Community Size for different Familiar Set Thresholds using the *Simple* Algorithm (MIT traces).

the iPAQ devices. There is a certain amount of people in a room, e.g. 5, and before entering the room in a certain order, the individuals have never been in contact before. If one waits long enough e.g. hours, days or weeks, they should all be in each others community. But when using *Simple* this is not the case.
The two persons that got into the room first will add each other to their familiar sets after some time as can be deduced from Algorithm 1. Since every person is in his/her own community set, the overlap with the other person's familiar set is maximal, and they add each other to their community. When the third person that entered the room is added as a familiar, the overlap of his/her familiar set with the community set of the first two persons is at the maximum again and therefore the third person is added to the community set of the first two persons. On the other hand the third person will not add the others to the community if $\lambda > 0.5$ in Equation 3.1 because their community set is already twice as big as the third person's community. The same is true for the fourth person if $\lambda > \frac{1}{3}$ and the fifth person if $\lambda > 0.25$. The suggested parameter in [24] is $\lambda = 0.6$.

The construction of communities thus depends on the order people include familiar peers and see each other. The approach in *k-Clique* may be less susceptible to that kind of instability because it needs a fixed number $k - 1$ of people in the community set belonging to the others familiar set in order to include a user in his/her community as Equation 2.3 shows. However, this criterion has the disadvantage of requiring a fixed number that overlaps both sets which results in the fact that people with big familiar sets being added to many communities and people with small familiar sets not being added that often.
The comparison of the community and the familiar sets, which are constructed based on two completely different criterion seems inadequate as a condition for constructing communities. We therefore changed the criterion and compare only the two familiar sets with each other. In order to require users to build up a familiar set before actually being added to someones community one can require an additional minimal familiar set size k. The resulting criterion to add a peer into one's community can be seen in Equation 3.2.

This modified community detection algorithm would not only solve the issue with the five people in the same room mentioned above but it also shows better results when applied to the MIT reality traces. We applied the original and the modified algorithm on the traces using $\lambda = 0.6$ and different familiar set adding thresholds and $k = 1$. In [24] one of the best results for *Simple* was obtained with a threshold between 150k and 250k seconds. In Figure 6.4 the resulting graph[14] of *Simple* for a threshold of 27h which corresponds to 226,8k seconds is shown.
The communities detected seem quite similar to the ones constructed by the *Modified Simple* algorithm shown in Figure 6.5. But when looking more closely one can see an important difference. In the original algorithm most of the connections are one-sided. This means that from a node's point of view, the communities may not look so nicely. Node 29 in Figure 6.4 for example has

[14]The graph was visualized with *Cytoscape* (http://www.cytoscape.org/).

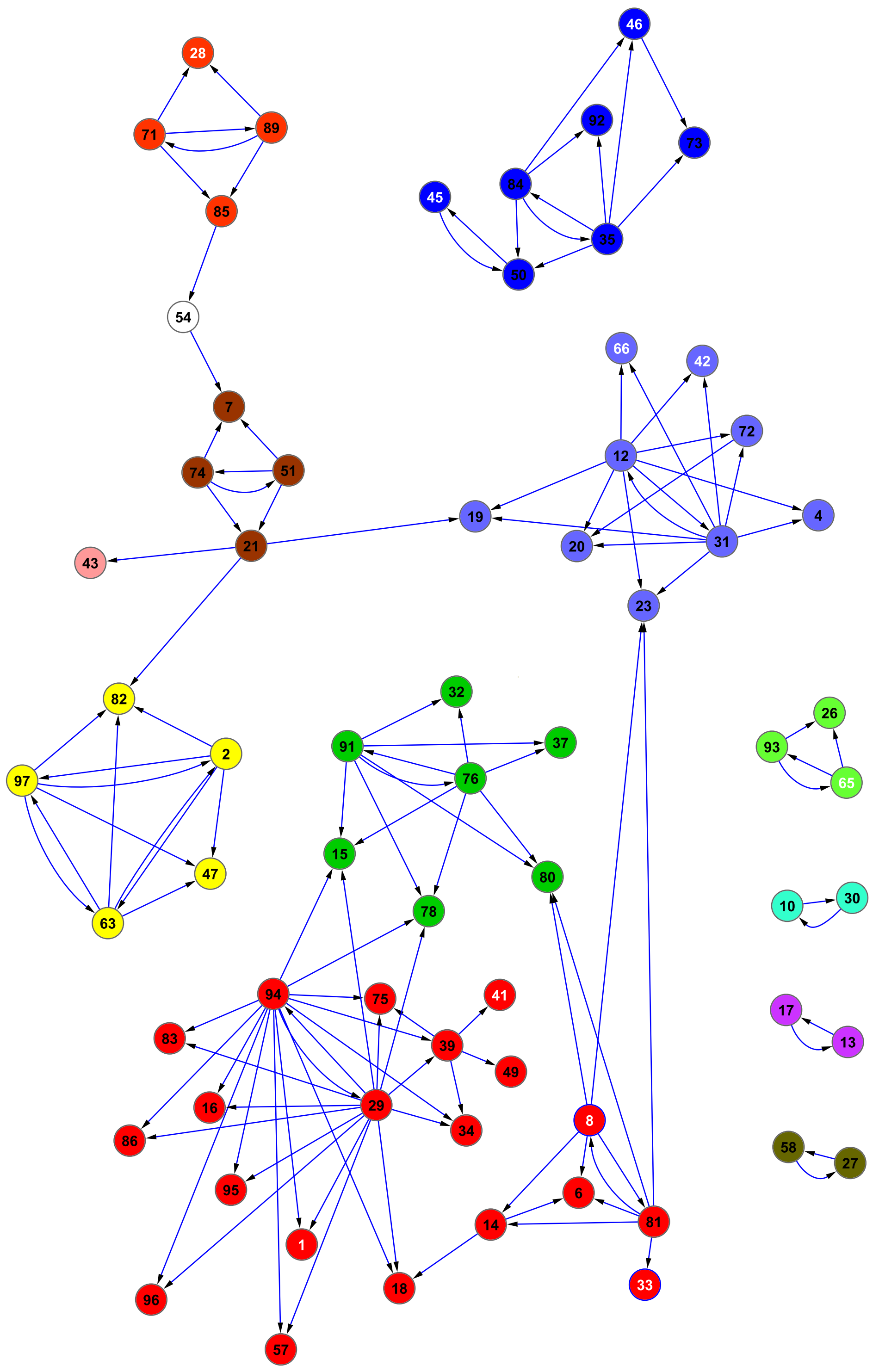


Figure 6.4.: Community Graph Constructed by the *Simple* Algorithm.

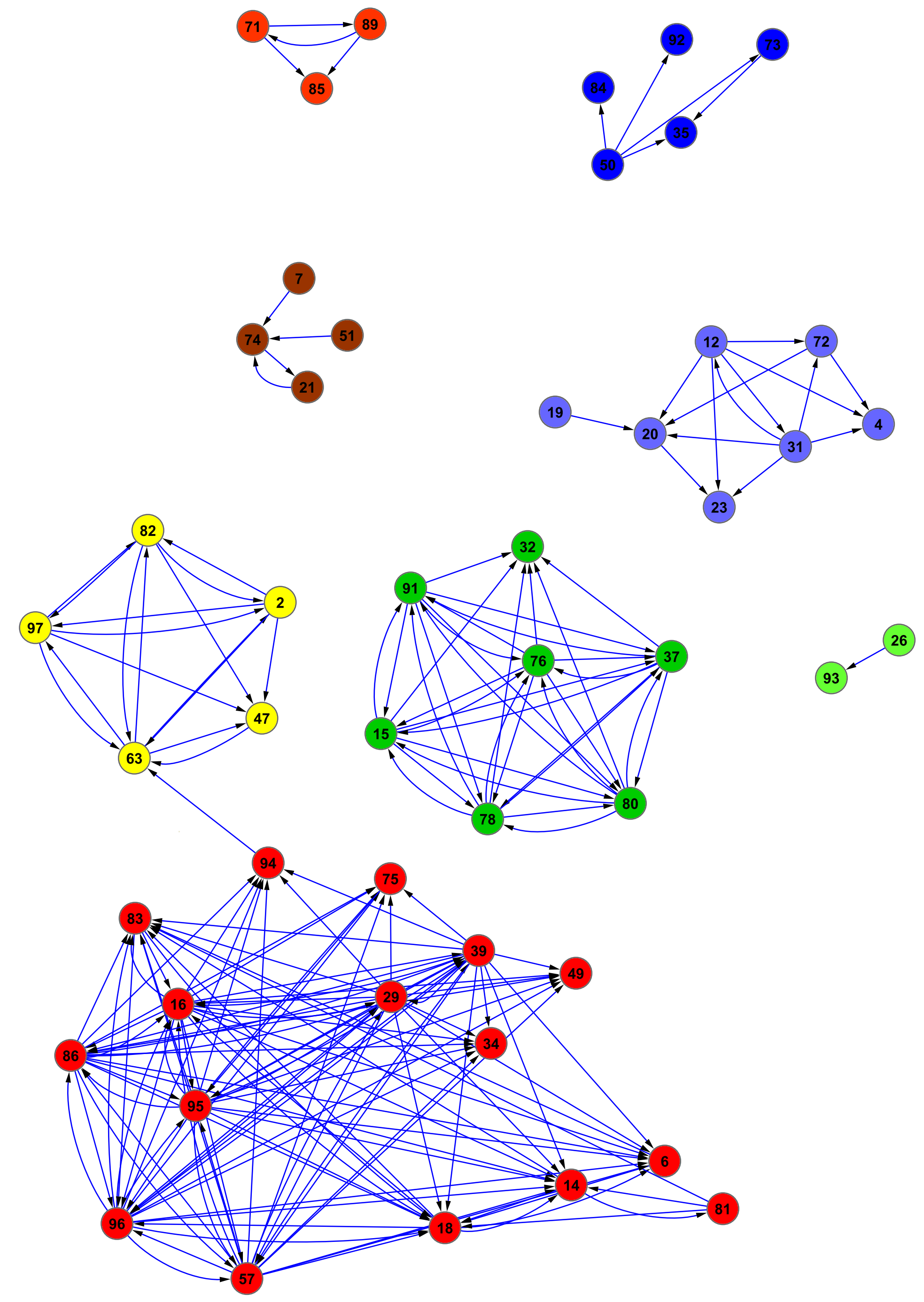


Figure 6.5.: Community Graph Constructed by the Modified Algorithm.

node 1, 15, 16, 18, 34, 39, 57, 75, 78, 83, 86, 94, 95 and 96 in its community, which corresponds to most of the community shown in red, but from the other node's perspective, e.g. node 34 or 75, their community is empty. This does not happen in the modified algorithm where the most of the connections are two-sided. In this example the average edges per node increase from 1.9 to 4.0 and the percentage of two-sided connection against the total amount of connections increases from 13.6% to 38.7%.

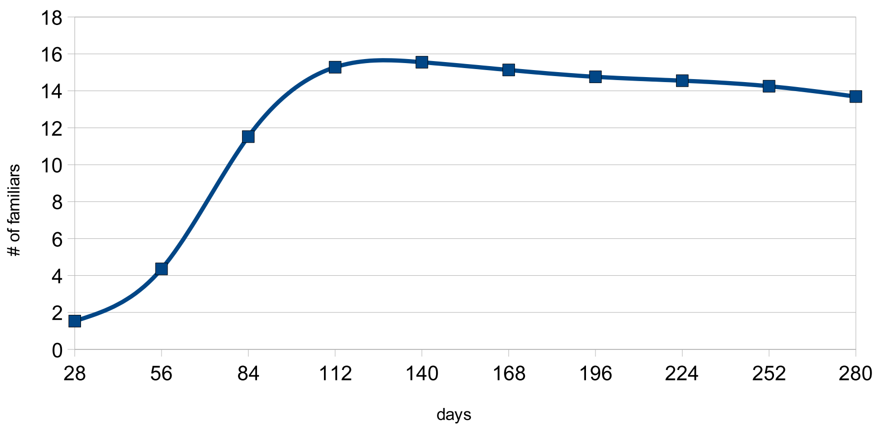
6.4.2. Dynamic Aging Extension

Although the community detection algorithm was significantly improved, the results presented in the last subsection show only an end result after a period of nine month in the MIT data set. Ideally, we would want a short initialization phase to build up a community which than can evolve over time. For this purpose an aging mechanism is required. The aging algorithm we propose is described in Subsection 3.5.4.

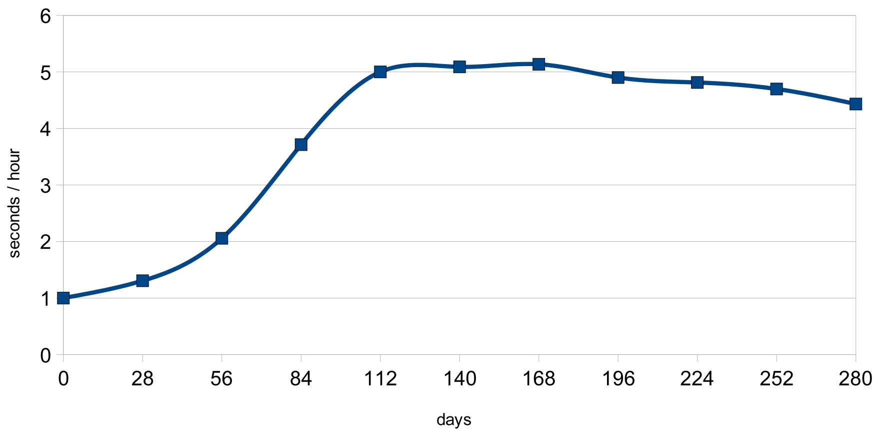
Before analyzing the results of the aging algorithm we should have a closer look at the MIT reality traces first. From the 96 people in the traces, 27 belong to the MIT Sloan business school whereas the rest are members of the MIT media lab. Although the two buildings are next to each other[15] there are no social connections between these groups as the analysis of the phone records show[16]. Except for the five freshmen, the subgroups of the media lab are more difficult to identify since they consist of different years of grad students and faculty members. A visualization of these groups can be found in Figure E.2.

The community detection algorithm using the aging mechanism needs a different set of parameters. We simulated several sets modifying different parameters and the outcome seems very stable which indicates that the dynamic behavior of the aging works quite well. A sample set of the fixed parameters can be found in Table E.20 and the initialization values of the dynamic parameters are listed in Table E.21. These where the parameters used to produce the results presented in the following.
In order to examine the effect of the community detection algorithm with the applied aging, the community graph as well as information of the dynamic parameters are analyzed every 4 weeks. This results in 11 snapshots of the community graph, whereas the first one is still during semester break.
The average familiar set size presented in Figure 6.6(a) shows an initial learning phase after which the value is stable at around 14. Note that the value is not only decreasing towards the end because of the applied aging mechanism but also because of a lower connection density at the end of the semester. The values show the same dynamic bahaviour as the aging speed and the familiar set adding threshold shown in Figure 6.6(b) and 6.6(c).

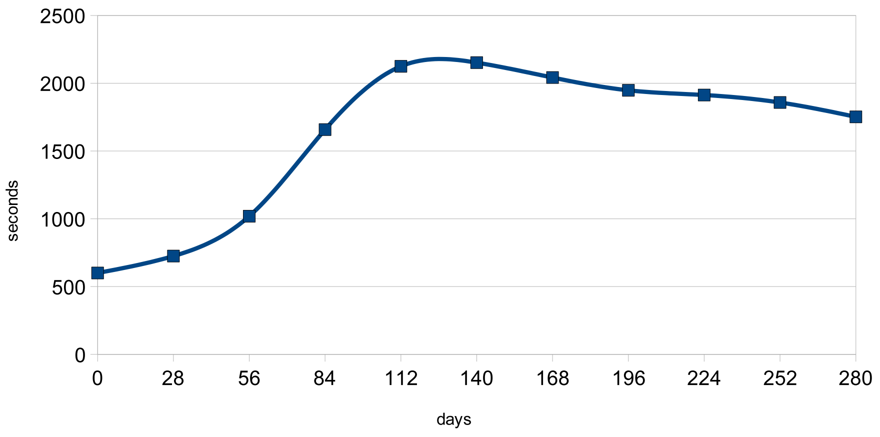
[15]See http://reality.media.mit.edu/dataset.php
[16]If two nodes have called each other, a social connection is assumed. The resulting graph can be found in Figure E.3.



(a) Average familiar set sizes over time.



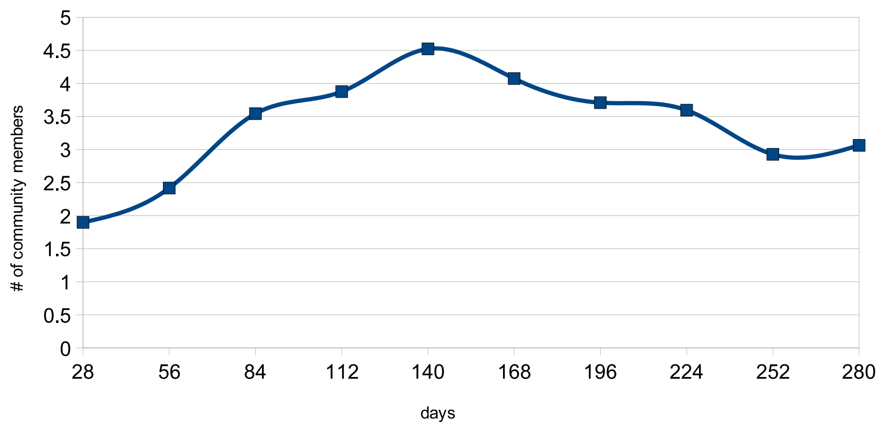
(b) Average aging speed over time.



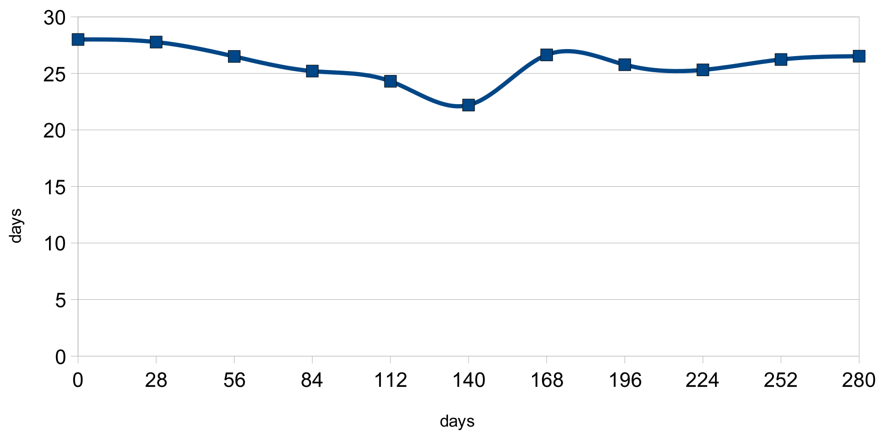
(c) Average adding threshold over time.

Figure 6.6.: Average familiar set size vs. aging parameters for the modified community detection algorithm with aging.

The average community size can be seen in Figure 6.7(a). The values around 3 to 4 are similar to the end result of the algorithm without aging. Figure 6.7(b) shows the community removal threshold, and as one would expect, it shows the opposite behavior compared to the community size.



(a) Average community set sizes over time.



(b) Average remove threshold over time.

Figure 6.7.: Average community sizes vs. aging parameters for the modified community detection algorithm with aging.

The course of the snapshots[17] of the community graphs also shows an interesting evolution. After the first four weeks, only a few people from the media lab are detected. The community is strongly interconnected and the center of it is a professor. In the second snapshot just around semester start, more media lab members are in the community. Additionally, four freshmen appear, forming a nice clique. The three next snapshots are all during semester so the media lab community has become bigger and the fifth freshmen completes the clique. On the other hand

[17]The figures can be found in Appendix E.4.3

the Sloan business school students have arrived forming a separate community. During that time, the communities look similar but the interconnections increase. After that time, the December break starts and the interconnections start decreasing which results in smaller community sizes. The last few snapshots belong to the spring semester. They have a bit less interconnections but the communities are similar. In particular, the separation between the Sloan business school and the MIT media lab can always be seen very clearly.

6.4.3. Security Implications of Community Detection

Applying community detection to PodNet rises new security relevant questions. There are two stages a peer can achieve with the applied algorithm, familiar or community member. Obviously one would not like peers to be able to cheat their way into our familiar or community sets.
The only way a peer is added to the familiar set is if he/she was in proximity for long enough. There is no way one can accelerate this process e.g. sending more frequent discovery packets is not effective since time is measured instead of counting the discovery messages. Once a peer is added the the familiar set one evaluates whether to add this users to the community as well. This evaluation can be fooled quite easily.
In the original *Simple* algorithm this can be achieved by only advertising the peer one is connected to in the familiar set. After being added to the the familiar set, the criterion to be added to the community will then always be true.
In the *k-Clique* algorithm the strategy to be added to another user's community is the opposite. One can add all encountered users to one's familiar set. This would also result in being added to everyones community set at least with a very high probability.
In the *Modified Simple* algorithm, an attacker can obtain community membership by receiving the familiar set of a victim in a first connection round and replay it in a second one.

The aging mechanism has a security drawback as well. If a stalker would constantly be present with a high number of identities, all these identities would be added to the familiar set. Since the familiar set size would increase, the aging would accelerate and legitimate users would be removed from the familiar set.

A detailed analysis of these issues is needed since the solution is still open.

7. Conclusion

In this chapter the main contributions are summarized, highlighting the *identity* management, *trust* metrics and *reputation* system which were introduced to PodNet. Furthermore, some general aspects of this work are discussed, like its *general applicability*, the implications and benefits a *hybrid network* architecture would introduce and the *privacy* concerns of such a content distribution mechanism. Finally, an outlook is given highlighting interesting areas for future work.

7.1. Summary

The goal of this thesis was to design a security concept for PodNet in order to assure the secure content dissemination among mobile users. To achieve this goal a proper user authentication was introduced, based on self generated *identities.* Users can now rate authors of content and asses their *reputation* by sharing the acquired opinion with others. The level of *trust* among users relies on the social ties such a friendship (built by *secure pairing*) or belonging to the same community (detected automatically). Additionally, a *spam control* mechanism prevents the flooding of content.
For the purpose of assessing the effectiveness of the proposed design, different security measures were simulated using real world traces and synthetic models. After observing satisfying results in the simulations, the security measures where integrated into the existing PodNet implementation. Finally, the quality of the *community detection* as well as the communication overhead of the security measures were evaluated.

The content distributed in PodNet can either originate from a central Internet server or from other users. In order to define security measures that best suit the different distribution requirements, our design introduces three channels, namely *open*, *restricted* and *closed.*
Traditional podcasts and Internet content may be published in *restricted* channels where authorship is limited to only a few people, whereas *closed* channels may be used in small groups which want to keep their information private (e.g. friends sharing holiday pictures). The most difficult and challenging distribution takes place in the *open* channel where content is user-generated and everyone is free to publish.

Securing content exchange in a completely distributed and open environment based on user reputation is as difficult as it is complex. The most basic but nevertheless crucial part of the design of the security measures comprises the unique generation of a user identity that cannot be imitated by others. By using the hash of a self created public key as *identity*, we can authenticate a user by challenging him or her for the private key. Additionally, users can become friends by consciously performing a *secure pairing* in which each user signs the other user's credentials as proof of the friendship.

As mentioned already, the identification of users is based on anonymous user-generated identities. Since user can generate multiple identities (sybil attack), we assess their *reputation* and *trust* as an indication of their quality as suppliers and authors which is not necessarily the same. The *reputation* measures the user's quality as author of content and is based on past experiences whereas trust defines a belief in the honesty of a user's future action. For our design, we assume to have an increased trust in users to which we are connected to, either socially (friends) or by the environment (community members) because we assume attackers to frequently change the identities to abandon bad reputation. A user wanting to change his or her identity is still free to do that but he/she cannot impersonate a reputable user and will lose all the gathered reputation and social connections when switching to the new identity. The author's signature appended to a content will always bind the authors's identity and thus implicitly his/her reputation to the generated content. A user that often changes his or her identity will neither have a good trust nor reputation value.

The *reputation* value for an author is built by assessing the quality of his or her published content. Since users meet in an opportunistic way, they cannot ask other users or a central database on demand and have to collect reputation information on their own.
In our simulations, we observed that calculating a reputation value only from direct experiences is inefficient since every user has to download bad content before reacting on it. Such a solution would thus be reactive, slow and content may still spread for weeks because of some users never rating received content. A good way to significantly increase the detection speed is the exchange of reputation information among users. It improves the detection in two ways, firstly, users may profit from global knowledge about an author and proactively block the author even without receiving any malicious content, secondly, even users that did not recognize the bad content as such will start blocking it and thus prevent its further dissemination. However, the problems with this approach are twofold, on the one hand it enables attackers to spread wrong information (lies) and thus influence the reputation in a negative way, while on the other hand, users themselves may rate content differently due to their taste.
In order to minimize the latter, we introduced a two level rating combining an objective and a subjective rating together. The *objective rating* ensures trusted content exchange by measuring the legitimacy of content which should be assessed similarly by the majority of users. The *subjective rating* reflects the users' satisfaction which may vary depending on a user's taste, culture, social situation or educational background.
In order to minimize the problem of lying user, they can be prevented from taking dispropor-

tionately high influence on the reputation by two measures. Firstly, by introducing a suggestion threshold of required objective ratings to ensure confidence in the rating, and secondly, by weighting all received ratings, with the user's trust value. Attackers that often change their identity will be viewed as strangers to whom only a minimum trust is assigned, whereas well known users, community members and friends will have much higher trust.

To the best of our knowledge, not much work has been done in the area of *community detection* for opportunistic networks. We therefore modified an existing lightweight algorithm that performed poorly and observed very promising results showing the detection of much better connected communities. We also added an *aging mechanism* which allows a much faster and more dynamic communities detection. Simulations with community weighted reputation values showed that when considering community information, the values of the suggestion threshold (i.e. the minimum number of recommendations required) can be doubled[1] compared to unweighted case without decreasing the detection speed. This implies that an attacker with a much lower trust weight than regular users would need much more effort in order to influence other user's reputation.

Since the *reputation* exchange suffers the serious drawback of requiring to downloaded content and consciously rate it before information can be exchanged, an attacker can flood the channel without fearing any bad reputation by frequently changing the identity. Therefore, we introduce a *spam control* mechanism that proactively limits the amount of content a user can download from the same author per day. This rate evolves with the reputation value of an author. Hence, only a limited amount of contents can be downloaded from completely unknown authors whereas the rate does not affect reputable authors. Additionally, the mechanism also serves as an incentive for the user to rate content in order to get more or less content from an author.

A real user's behavior is difficult to model because when it comes to rating content, it may depend on many various properties such as taste, social situation, culture or educational background which may sometimes even not be rational. Therefore, a real deployment is needed in order to examine the applied security measures under realistic conditions.

7.2. Discussion

In this section we take a closer look at some of the approaches used in this work and discuss how general their application is in order to be used on other types of networks and to solve other problems. Since PodNet is originally designed for a hybrid network, i.e. an opportunistic network combined with a fixed infrastructure, the benefits of such networks are examined as

[1]Recommendations of community members are counted as multiple suggestions from strangers in order to give them a higher weight.

well. In the last part some privacy issues are covered, and how the proposed security measures deal with them.

7.2.1. General Applicability

In this work we treat several different concepts which are interesting beyond the borders of PodNet and opportunistic networks. The most important ones are *Identity Management*, *Trust Metrics*, and *Reputation*.

Identity Management: The management of identities in an opportunistic network is a difficult task. There are two main challenges when entities are able to build their own credentials. They consist of making identity theft impossible and inhibiting sybil attacks. The first, i.e. the protection of a user's identity, has been solved by asymmetric cryptography. As long as an identity is bound to a public key and as long as the user manages to keep the corresponding private key secret, the identity can be verified assuming the cryptographic algorithms are secure.

The second and more challenging problem when lacking a central authority is avoiding the generation of many identities by a single user (sybil attack). While a true one-to-one, once in a lifetime binding between an identity and an individual is already challenging when having a central infrastructure at one's disposal, it becomes impossible in a fully self organized environment. Although occasional changes of a user's identity would not be terrible, the generation of a large amount of simultaneous identities becomes a problem since it opens the possibility for many different attacks, e.g. increasing one's influence in a distributed rating system.

This work tackles this problem from a social angle by establishing certain *Trust Metrics*. This has been done in other areas already, e.g. in PGP [7] chains of trust are built. While chains usually require routing or at least the availability of all the certificates to complete a chain, the approach we used is more general since it has no such constraints and can thus be applied in a large number of different types of networks as discussed in the next paragraph.

Trust Metrics: In order to establish trustfulness of a user we assume, similar to PGP, the transitivity of trust to a certain degree. Whereas in PGP a user is either trusted or not, depending on whether a chain is found, we distinguish between different trust levels by classifying social ties in different categories. Although we currently just have five different categories (and associated trust levels), namely *friends* (highest trust), *friend circle member* (high trust, somewhat below friends), *community member* (low trust), *familiar* (lower trust but close to community members) and *stranger* (minimal or no trust at all), this can be extended to provide a more fine grained scale of different trust levels. One can for example trust a user with three common friends more than some distant member in our friend circle, with whom we have no common friend with at all. No matter how fine grained the different trust levels, or how complex the algorithms are, the basic concept remains valid for any distributed network. Additionally, this concept cannot only be used to calculate the probability of an identity belonging to a trustworthy user, but also other areas, e.g. for routing and forwarding.

Another way to establish trust is to calculate a similarity between two users. This can be done in several ways, e.g. by comparing how much of the same or at least similar subscribed channels are available. A better way would be to compare how similarly two individuals rate content. This has several advantages over comparing the similarity of available channels. Firstly, a user has to rate consciously, which increases the probability of dealing with a valid identity and secondly the ratings have to be similar which not only assures a comparable taste, but also suggests the rating is not being randomly generated.
This way of assessing trust can be easily combined with the social network approach which makes it a good add-on. The drawback is that it is bound to a system where the offered services can be rated, like providing content in this case.

Reputation: The reputation framework used in this work can be applied to any other content exchange protocol or system where users provide a service. The reason lies in the general nature of PodNet which is based on the distinction of the author and the supplier of a content. While other systems require direct connections between the user being rated and the one rating, PodNet does not guarantee such a connection, it is actually highly unlikely to exist at all. We build up a reputation about a certain user using our own and our surrounding's opinion without any restrictions on that user. It does not matter whether the user is somebody who has never been met, a member of the same community, a friend or even oneself, the system works transparently for everybody. Another reason for this transparency is that we do not assume a trustworthy user to produce content we necessarily like, thus separating trust and reputation. This makes it very flexible and protocol independent.

7.2.2. Hybrid Networks

The original PodNet concept is based on a hybrid network consisting of an opportunistic part which allows users to exchange content among themselves and a limited amount of gateways which allow the distribution of Internet content. Although the security measures proposed in this work are mainly based on the opportunistic part of PodNet most of them could benefit significantly from the hybrid architecture. Specifically, the creation of a secure identity with a higher resilience to sybil attacks can be achieved with a registration process as described in Section 3.3.2 under *Registration*. Such a registered identity could also allow the verification of additional feature like age which would be useful to limit channels providing explicit content for the view of adults only.

Another part of the security measures which could benefit from the hybrid architecture is the reputation system. It not only allows the usage of a global blacklist as described in Subsection 3.4.3, but also the caching or collecting of user generated ratings in order to receive a more detailed opinion about an author and spread it among users. Since the infrastructure part can communicate over a different medium, e.g. the Internet, the reputation of an author can be spread much faster and also in areas the content has not yet reached.

7.2.3. Anonymity and Privacy

One of the main issues when designing a security concept for PodNet was to make sure we can hold users accountable for their actions. At the same time a user would like to stay anonymous, so no other user should be able to link the real person to the user's content or rating. This becomes obvious when thinking what could happen if a boss discovers his or her content is being badly rated by his or her secretary. Our Secure PodNet guarantees this kind of anonymity by using the hash of the public key as the identity. Although an identity can be held accountable for what the corresponding user does, either by blocking or rating, there is no way of identifying the person behind the identity. As mentioned before, this is generally wanted, but it could be problematic when content is illegal and the author should be identified and possibly sued. This problem could be solved by the registration explained in the previous subsection.

Another privacy issue arises when publishing the owner of a channel. In general, the owner should have the option to stay anonymous, since he or she might not want to be linked to the content other users publish in the channel. A similar issue exists in the closed channel, where the different members might not want to be linked to the channel itself or to other members in the channel. Both issues are taken care of in the proposed security measures. A channel is represented by its own credentials and only optionally linked to the owners ID and the member list of a closed channel is only shared among themselves.

There is one last privacy concern which is not considered by the current security design. It allows for a low level attack and comprises the tracking of an ID. Discovery messages are sent out regularly and since they contain the user ID, they can be used to follow a person. This can be done independently of the fact that the user ID is just an anonymous hash which does not allow the explicit identification of the person being tracked. This is a very general problem beyond the scope of PodNet and this work and remains open.

7.3. Outlook

In this section we present suggestions for further investigations and improvements to the current PodNet system. Because of the wide variety of topics in this work, we divided them into the following subjects: Podcast, Rating, Identity and Evaluation. The implementation specific improvements can be found in Section 5.7.

Podcast: The current podcasting system comprises several limitations and drawbacks. Some of them may be reduced by:

- Redesign of the discovery channel in order to improve visibility and relevance of its content by limiting the exchanged channels, introducing channel aging and define channel rating as explained in Section 5.6. Besides that, the exchanged informations may be extended to include channel descriptions which could give users insights whether to subscribe to a channel or not.
- Refining and applying channel policies introduced in Subsection 3.4.1 may restrict the channel access for certain persons. A user may require a certificate or signature which states that he/she fulfills certain properties, such as e.g. being of legal age, which are required to view contents from a channel.
- Introduction of revocation lists which include episode IDs that should automatically be deleted by every user in the channel in order to remove content without blocking its author.
- Deployment of a Bloom Filter for episodes as has been done for channels in the original implementation [3] in order to reduce the transmission overhead and thus decrease the synchronization time.
- Introduction of channel tags similar than in [40], [41] which describe the content of a channel. Instead of human readable text, keywords may be used in order to evaluate similarities between channels which may be used when assigning user similarities (see *Rating* below).
- The integration of incentives or game-theoretic approaches may additionally increase reliability and fairness of the content dissemination process.

Rating: The author reputation presented in Section 3.6 may be extended by the following:

- Introducing similarities between received recommendations as in [9] or [16]. We currently rate the weighting based on the environment which may be reasonable for objective ratings but inaccurate for subjective ratings because of different tastes inside the community. Additionally the similarity of two persons' local channel list can be calculated and be used as an indication for similar taste.
- Exploiting the similarities of channels may additionally be useful when only a few author information is available in a specific channel but plenty of it in a very similar one. It may then be reasonable to derive the reputation from these related channels [9].
- The effect of subjective rating may be increased. Currently, we primarily use the objective rating in order to block misbehaving peers and use the subjective ratings only to vary the limit in the rate limitation. It may be advantageous to allow manual combination of both ratings in a way that all objectively rated content may be considered if only few content is available and, additionally, subjective rating is enabled to set a basic rating requirement when many contents can be found.

- Make the rate limitation dependent on environment. In an environment of only a few people, one may allow more content downloads and be less restrictive than in an environment of many people where, due to the potential high amount of contents, also legitimate data may be viewed as spam.
- Assigning a confidence value [42][2] to the received recommendations may help assessing the trust of rating values in the combination process[3].

Identity:

- Find mechanisms which efficiently prevent sybil attacks in opportunistic networks without requiring any central authorities.
- Instead of the the IPv4 address, a mobile IPv6 address which could also serve as identity and thus prevent ambiguities and address collisions in SyncList and SyncHistory.
- More Investigations towards identity and certificate revocation is needed.

Evaluation:

- A real world deployment is needed in order examine social connections and evaluate the effectiveness of the community detection as well as the users' rating behavior under realistic conditions.
- The proposed community detection algorithm should be analyzed with real world data that reveal the social connections. Ideally, one may try to make the community parameters dependent on the environment.
- It may also be reasonable to make the recommendation weights and suggestion threshold dependent on the environment, i.e. the amount of people in one's community.
- Evaluate the security implications of community detection as mentioned in Subsection 6.4.3 and make it hard to influence an other user's view of his/her community.

[2]Confidence value: the number of people that have provided a specific rating.
[3]See Section 3.6.3.

Appendices

A. ‘The One’ Simulator

As mentioned in Section 4.3, we had to modify ‘The ONE’ simulator in order to use it for our purpose. The main problem consisted in the simulator being connection and message oriented. In order to simulate the spreading of content the hosts needed to be able to save data. For this reason a ‘HostBody’ module was inserted into each host represented by the ‘DTNHost’ module. This module takes care of all the abilities a host requires in our scenarios.

In order to make use of the new functionality in the hosts we needed the ability to configure them. For this reason a ‘ConfigurationEvent’ was created. In order read out configurations from a file we modified the ‘StandardEventReader’. It can now interpret a commands in the following format: ‘CONF <host ID> <command> <argument>’. For the available commands, the code should be consulted.
Since usually the configuration such as the generation of content should not be read from a file but be done periodically, a ‘ConfigEventGenerator’ module was created. The configuration event generator can be configured via the settings file. In order to handle all the new settings correctly the ‘Settings’ module had to undergo some small changes. The available settings can be found at the end of this chapter.

The hosts also need the ability to synchronize content and other information. For this reason a ‘SynchronizationEvent’ was introduced. Since the synchronization events should be triggered by a connection between two hosts, a ‘ConnReactiveEventGen’ module was implemented. As the name implies it has the ability to generate events whenever two hosts connect or disconnect.

In order for the configuration event generator to perform certain tasks regularly and for the connection reactive event generator to detect connections and disconnections of hosts, they have to implement the ‘UpdateListener’ and the ‘ConnectionListener’ interface. Since those interface where originally implemented for the reports, a hack was needed to make them available to the event generators. For this reason the ‘SimScenario’ as well as the ‘EventQueueHandler’ module were slightly modified.

In order to output all the simulated data some new reports were created. The most frequently used were the ‘BLContentReport’ which outputs content spreading data that can be read by Matlab, and the ‘CommunityReport’ which outputs community related information including a community graph. For the functionality of the other reports added, namely the ‘ContentReport’, the 'ContentSpreadingReport', the 'ConnectivityGraphReport' and the 'SpreadingGraphReport', the code should be consulted.

Settings: There are two new event generators that can be enabled in the settings. The 'ConnReactiveEventGen' can only be enabled but does not take any parameters. The 'ConfigEventGenerator' on the other hand takes parameters to specify the content and spam generation intervals to configure the different roles of users and their behavior. For more information one can look into a settings file or into the code of the 'ConfigEventGenerator' module.

B. MIT Data Set

B.1. Data Set Properties

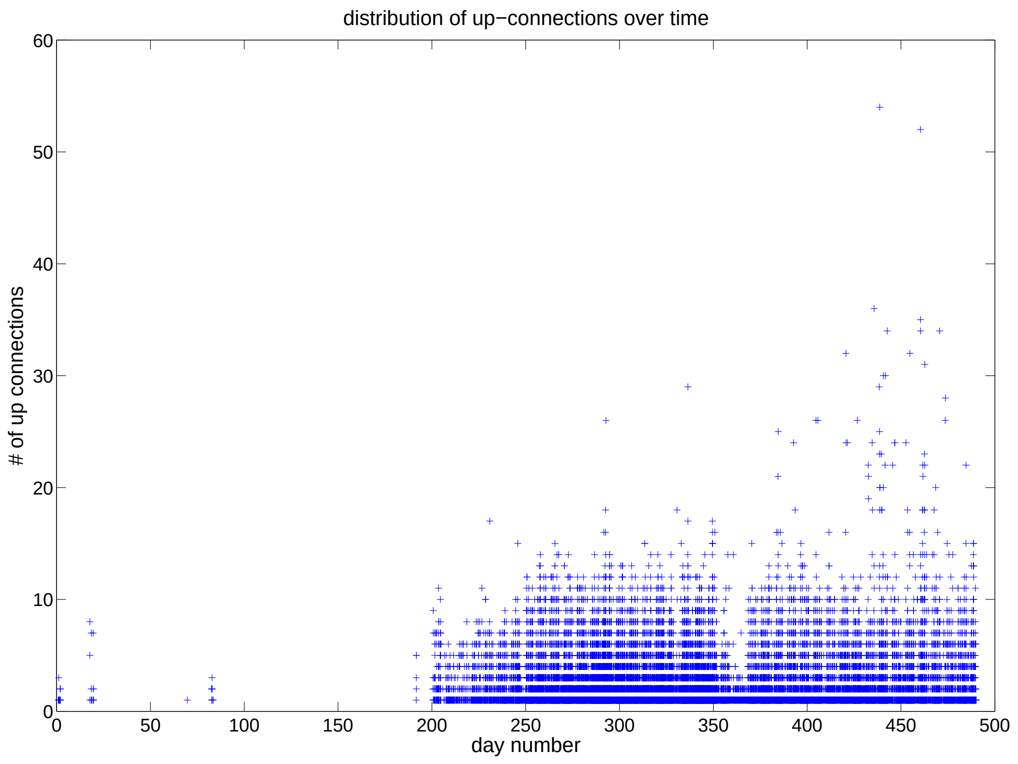


Figure B.1.: Number of Bluetooth Up Connections in the MIT Data Set.

The collected data comprises a lot of detailed information from which we just used the Bluetooth connection data as trace information.
The data set consist of 97 persons from which 8 persons have never seen any other contact, these are persons with IDs: 25, 48, 52, 53, 55, 64, 67, 87. Person 25 is additionally also never seen by any other person, i.e. person 25 is isolated.
The total simulation time is 491 days. The first entry in the data collection is made at the 1^{th} of January 2004 and the last at the 5^{th} of May 2005. At day 360 there was Christmas eve where

No.	Month	# Days	Ref. Days
1.	Jan' 04	31	0
2.	Feb' 04	29	31
3.	Mar' 04	31	60
4.	Apr' 04	30	91
5.	May' 04	31	121
6.	Jun' 04	30	152
7.	Jul' 04	31	182
8.	Aug' 04	31	213
9.	Sep' 04	30	244
10.	Oct' 04	31	274
11.	Nov' 04	30	305
12.	Dec' 04	31	335
13.	Jan' 05	31	366
14.	Feb' 05	28	397
15.	Mar' 05	31	425
16.	Apr' 05	30	456
17.	May' 05	31	486

Table B.1.: Conversion Table: Date - Day Number.

almost no activity was measured. The Semester of the academic year 2004/05 at MIT started at around day 251 (first day of class 8.September) [1]. The day - date conversion can be computed with the help of Table B.1. In order to calculate the original day number to a specific date, go to the corresponding month in Table B.1 and add the number of days in the date to the reference number.

B.1.1. Conversion Table Date - Day

However, Figure B.1 shows that the first 200 days have only a few Bluetooth Up Connections. In a first simulation round, we therefore discarded them and re-numbered the days and started counting from original day 200, i.e. Monday July 19, 2004. In order to find the correct date, always subtract 200 days in Table B.1.

In Figure B.2, we see that the quality of the individual weeks can vary a lot which motivated us to classify the weeks and perform a second simulation round.

[1]http://web.mit.edu/registrar/www/calendar0405.html

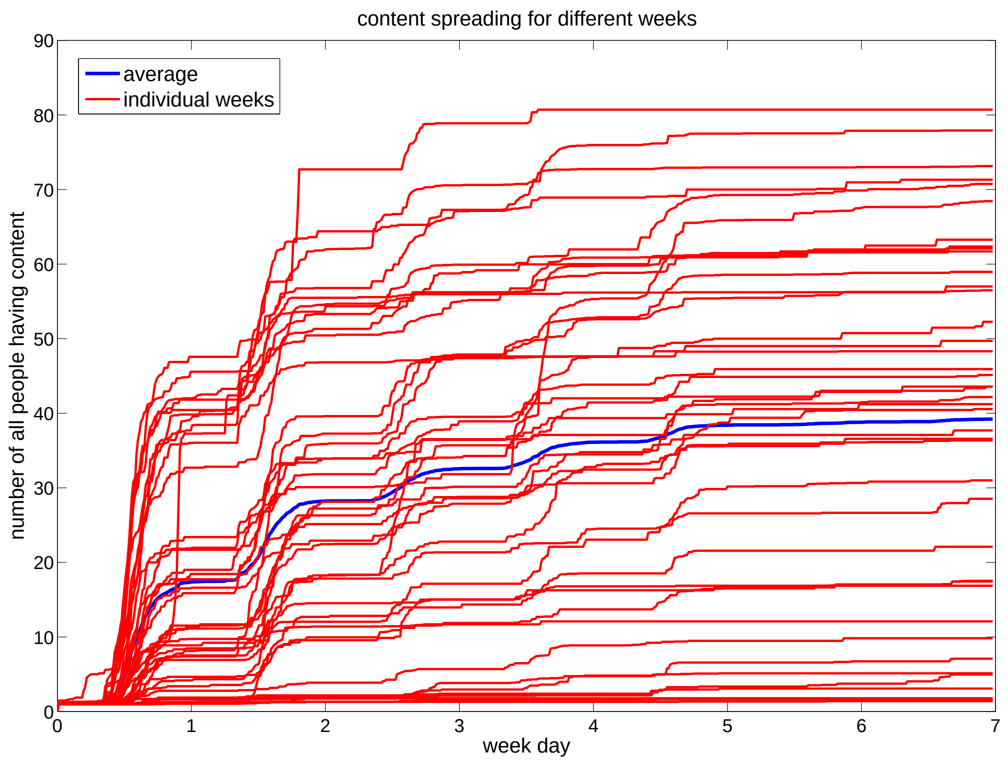


Figure B.2.: Spreading Performance for Different Weeks in the Unclassified MIT Data Set.

B.2. Classification Process

Because of the week quality difference, we classified the weeks in a second simulation round into good, average and bad weeks. The classification was performed by hand by observing the spreading performance of one content sent per week for all users and splitting the weeks based on the average number of people having the content. The limits are set as follows (a being the number of people having the content):

Unfortunately, after having discarded the first 200 days, the dataset only consisted of 42 week.

Criterion	Action	Clustered Weeks	Total Weeks
$a < 5$:	discard week	1-3, 24	4
$5 < a < 24$:	bad week	4- 8, 23, 25, 36, 38, 39	10
$25 < a < 47$:	average week	26, 27, 31-35, 37, 40, 41	10
$47 < a$:	good week	9 - 22, 28-30	17

Table B.2.: Classification of Weeks with Similar Quality

Therefore, we had to repeat repeat the usable weeks several times in order to have 30 weeks per scenario. The classification is shown in the following table:

Classification	Weeks
Good	9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 28, 29, 30, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 28, 29
Bad	4, 5, 6, 7, 8, 23, 25, 36, 38, 39, 4, 5, 6, 7, 8, 23, 25, 36, 38, 39, 4, 5, 6, 7, 8, 23, 25, 36, 38, 38, 39
Average	26, 27, 31, 32, 33, 34, 35, 37, 40, 41, 26, 27, 31, 32, 33, 34, 35, 37, 40, 41, 26, 27, 31, 32, 33, 34, 35, 37, 40, 41

Table B.3.: Classified Week Sets

As Figure B.3 shows, the average of all 'average weeks' correspond approximately to the average spreading curve of the unclassified weeks in the first simulation round. It is a little bit higher because of less people being active in the average weeks.

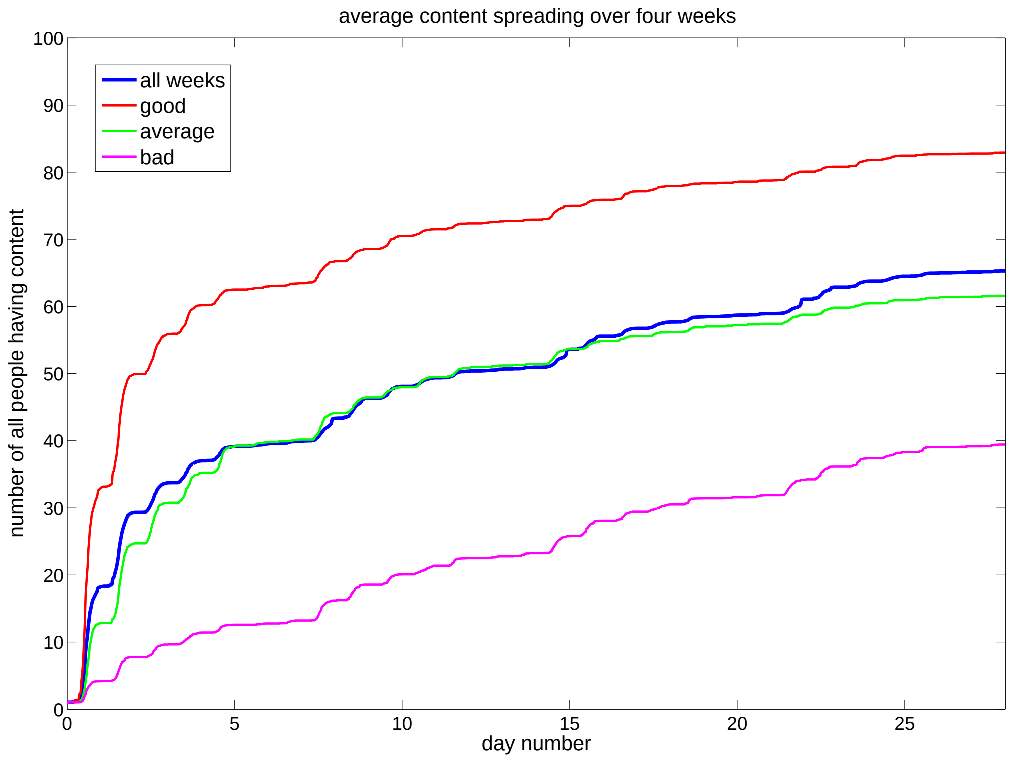


Figure B.3.: Average Spreading Performance of Unclassified Weeks Compared to Good, Average and Bad Classified Weeks.

B.2.1. Source Information

oid	survey position	survey neighborhood	survey hours	survey regular
1	-	-	-	-
10	mlgrad	Fresh Pond*	11am-9pm*	very
11	1styeargrad	Central	11am - 8pm	somewhat
19	sloan	-	-	-
20	sloan	-	-	-
30	5thyear	Central	11am - 8pm	not at all
31	sloan	-	-	-
39	student	Porter	9am - 5pm	very
40	sloan	Central	11am - 8pm	somewhat
49	6thyeargrad	MIT	11am - 8pm	very
50	sloan	Boston	9am - 5pm	very
59	student	MIT	9am - 5pm	somewhat
60	8thyear	-	-	-
67	sloan	-	-	-
68	mlfrosh	MIT	no schedule*	not at all
69	newgrad	-	-	-
70	masfrosh	MIT	4pm-8pm*	somehwat
77	sloan	MIT	11am - 8pm	somewhat
78	student	Boston	11am - 8pm	somewhat
79	sloan	-	-	-
88	staff	Brookline	9am - 5pm	very
89	newgrad	MIT	11am - 8pm	very
97	newgrad	Boston	11am - 8pm	somewhat

Table B.4.: Source Information Information of 23 Randomly Selected Nodes: *These nodes were randomly selected (same seed) in all simulations with the MIT data set.*

C. Simulation

C.1. Non-Cooperation

Percentage	MIT Unclassified	MIT Good	MIT Bad	MIT Average	Haggle 24h
0	100%	100%	100%	100%	100%
10	99.18%	99.43%	96.44%	99.26%	99.59%
20	98.73%	98.49%	91.85%	98.61%	99.18%
30	97.89%	98.05%	87.90%	97.22%	98.62%
40	94.77%	97.11%	85.46%	95.56%	97.34%
50	93.56%	96.86%	81.67%	93.35%	96.76%
60	91.76%	96.02%	76.16%	89.53%	95.65%
70	86.99%	94.38%	67.58%	83.29%	93.03%
80	79.00%	89.71%	56.33%	70.31%	90.72%
90	57.31%	66.64%	36.89%	42.34%	85.99%

Table C.1.: Non-Cooperation in Varying Environments: *The average spreading performance of one content after exchanging it for four weeks.*

C.1.1. Unclassified MIT Set

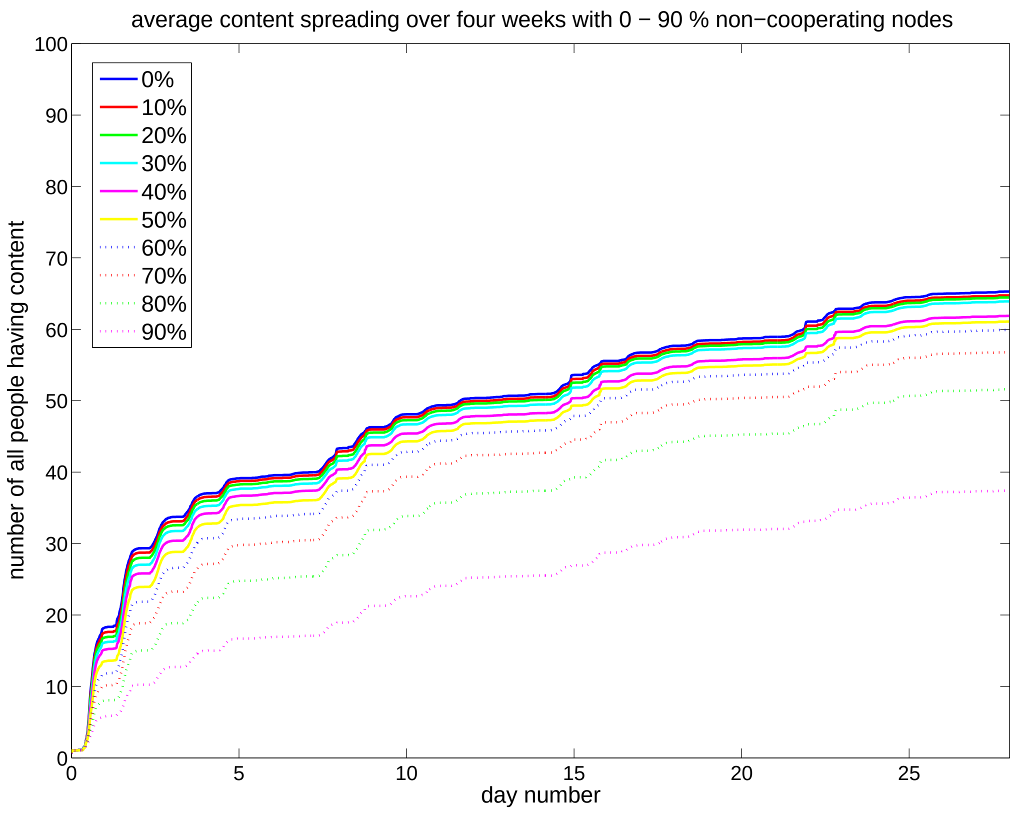


Figure C.1.: Non-Cooperation in Unclassified MIT Weeks: *Average spreading performance of one content generated at day 0, observed over a period of four weeks.*

C.1.2. Classified MIT Set

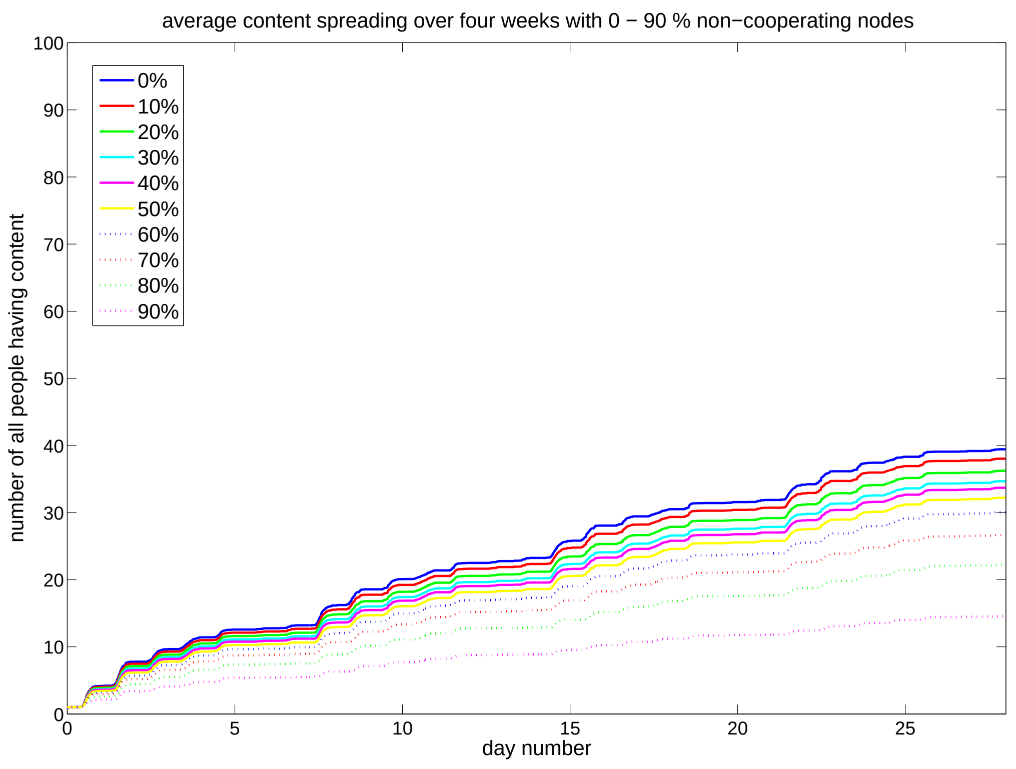


Figure C.2.: Non-Cooperation in Bad Classified MIT Weeks: *Average spreading performance of one content generated at day 0, observed over a period of four weeks.*

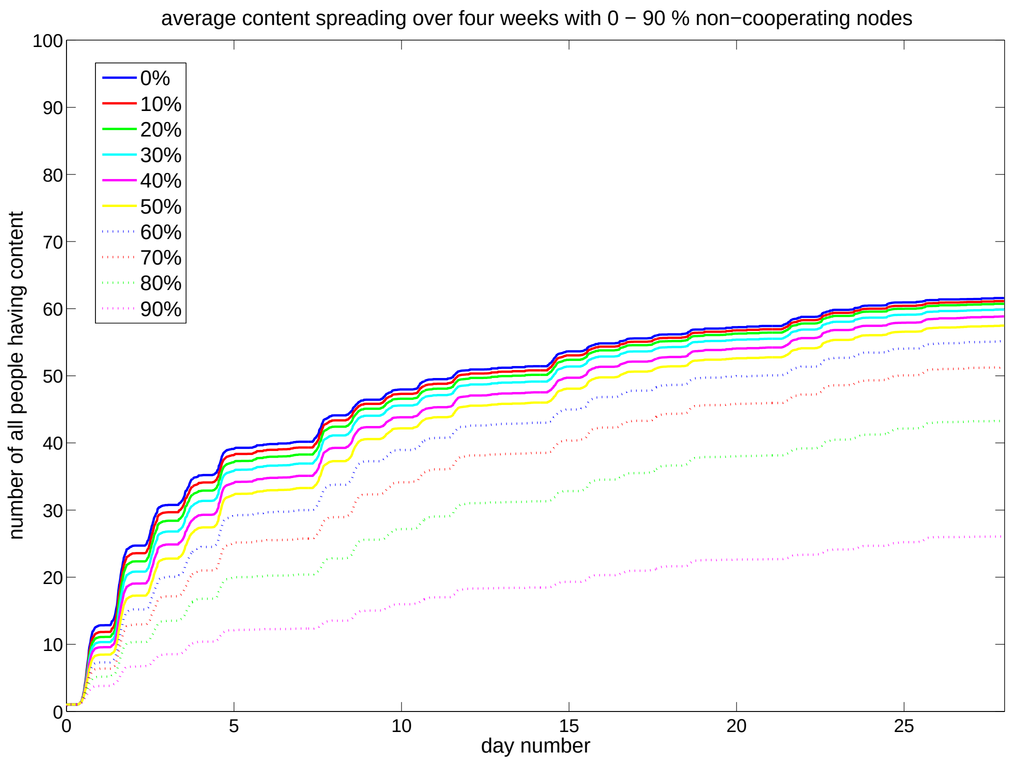


Figure C.3.: Non-Cooperation in Average Classified MIT Weeks: *Average spreading performance of one content generated at day 0, observed over a period of four weeks.*

C.1.3. Haggle

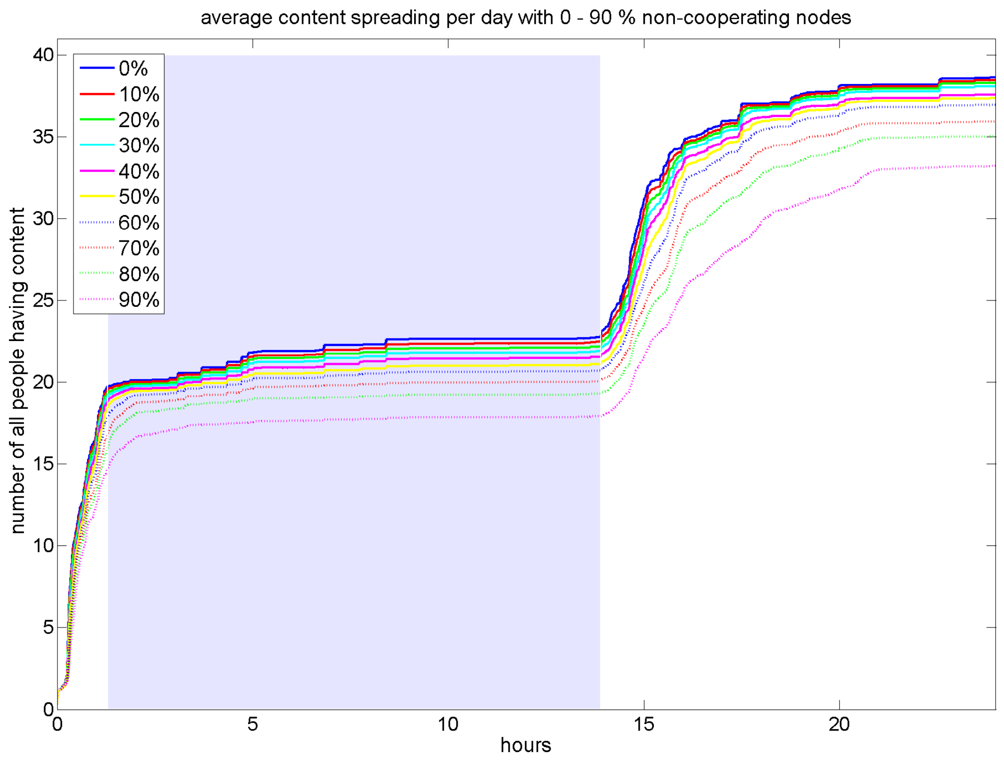


Figure C.4.: Non-Cooperation in Haggle Data Set: *Average spreading performance of one content generated at day 0, observed over a period of 24 hours. The shaded regions mark an 8 hours inactivity period during night.*

C.1.4. Synthetic Traces

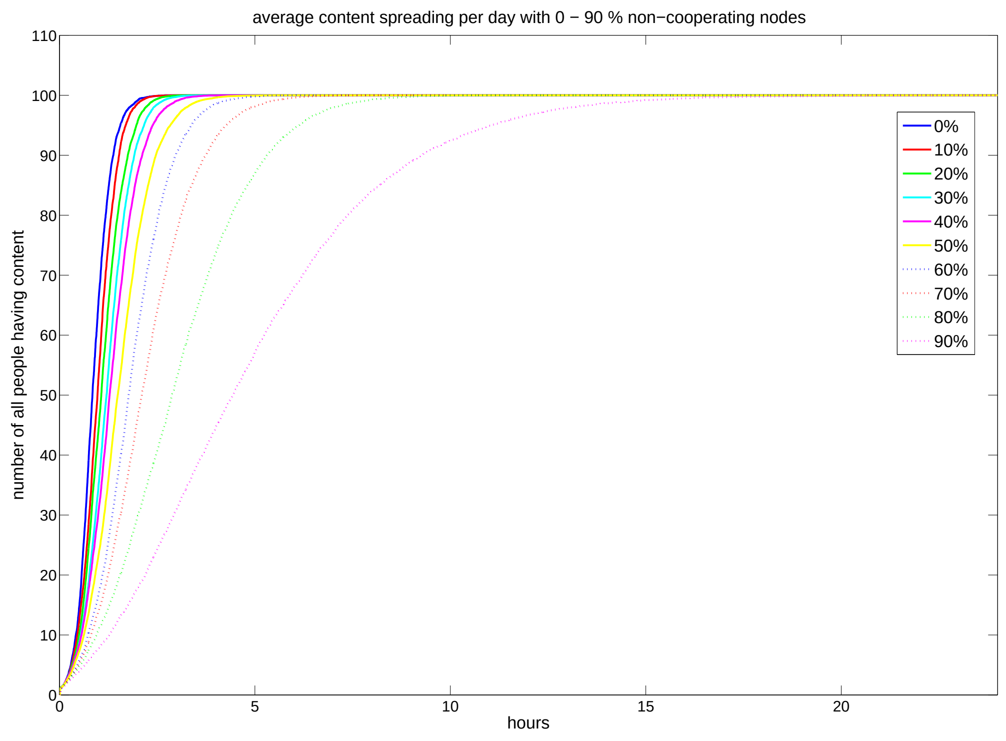


Figure C.5.: Non-Cooperation in Random Waypoint Model: *Average spreading performance of one content generated at day 0, observed over a period of 24 hours. The corresponding parameters are specified in Section 4.2.*

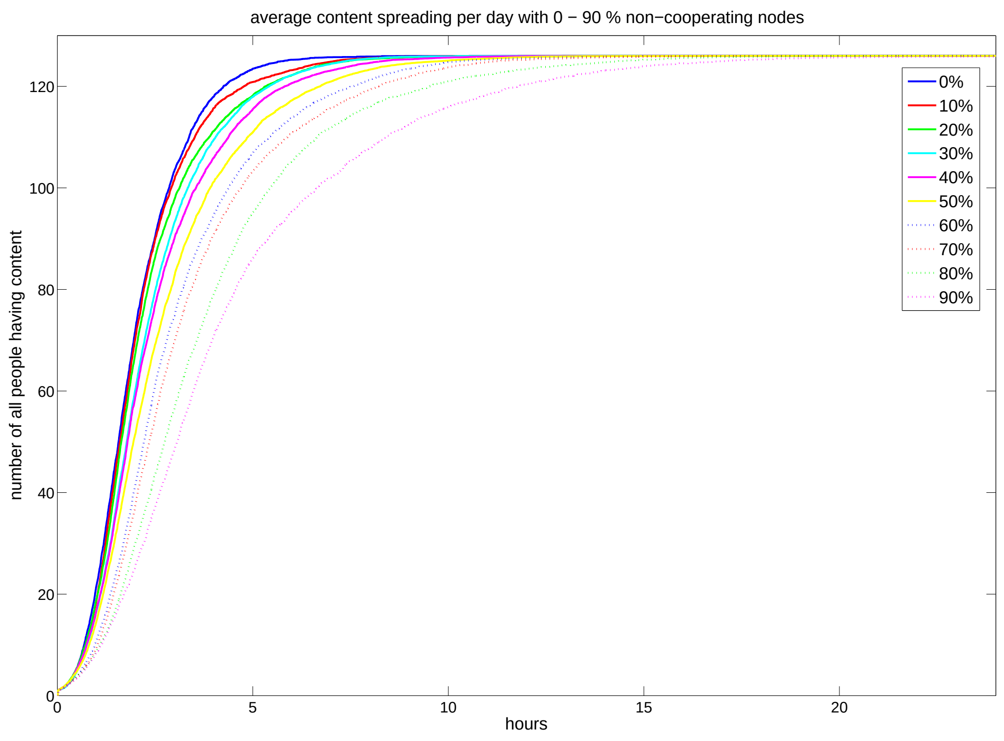


Figure C.6.: Non-Cooperation in Helsinki Model: *Average spreading performance of one content generated at day 0, observed over a period of 24 hours. The corresponding parameters are specified in Section 4.2.*

C.1.5. Comparison Haggle - Synthetic Traces

When comparing the synthetic traces of in Figures C.5 and C.6 with the high density real world traces from haggle in Figure C.4, we observe that the content dissemination speed is similar if neglecting 8h of inactive night shift as Table C.2 shows. There are two numbers for each simulations in the Haggle set. The number in brackets shows the real time including the 8 hours night shift, the other number shows the time when subtracting the night shift. In particular the times from the Helsinki model are very close the real world Haggle set.

Percentage	Haggle: 40%	60%	80%	Helsinki: 40%	60%	80%
0	3150	6500 (51850)	10500 (55850)	5640	7480	10430
10	3300	6750 (52100)	10850 (56200)	5780	7680	10670
20	3500	6900 (52250)	11100 (56450)	5900	7960	11410
30	3600	7300 (52650)	11500 (56850)	6400	8630	12210
40	3650	7400 (52750)	12100 (57450)	6470	8920	13060
50	3800	7700 (53050)	12400 (57750)	7010	9860	14330
60	3950	8050 (53400)	14250 (59600)	7980	10910	16030
70	4300	8950 (54300)	17600 (62950)	8480	11670	17210
80	4650	9750 (55100)	20250 (65600)	9890	13880	20010
90	6100	11900 (57250)	30300 (75650)	11060	15530	24370

Table C.2.: Non-Cooperation with Haggle Traces and Helsinki Model: *This table shows the times in seconds until reaching 40%, 60% and 80% of all people in the Haggle and Helsinki Set for different levels of non-cooperation.*

Percentage	Random Waypoint: 40%	60%	80%
0	2710	3400	4260
10	3110	3850	4770
20	3390	4290	5360
30	3850	4790	5950
40	4130	5160	6500
50	4740	6010	7600
60	5660	7110	9120
70	6640	8570	11190
80	8890	11880	15850
90	13140	18850	26570

Table C.3.: Non-Cooperation in Random Waypoint Model: *This table shows the times in seconds until reaching 40%, 60% and 80% of all people in Random Waypoint model for different levels of non-cooperation.*

C.2. Moderator Blacklisting

C.2.0.1. Unclassified MIT

	Unclassified	
time	having	providing
1 week	68.18%	34.06%
2 weeks	70.42%	14.42%
4 weeks	69.77%	4.33%
8 weeks	69.29%	1.69%
16 weeks	68.57%	1.04%

Table C.4.: Moderator Blacklists: *The table lists the percentages of people providing and having the first content in unclassified MIT weeks. The week times are all relative to the spam recognition of the (only one) moderator. The percentages are obtained with respect to the people receiving the content with normal spreading.*

classification	time	tot. having	max. providing (of all 96)
Unclassified	58d	70.29%	31.11%

Table C.5.: Moderator Blacklists in Unclassified MIT Weeks: *The table lists the percentages of people providing and having the first content in the Unclassified MIT weeks. The week times are all relative to the spam recognition of the (only one) moderator. The percentages are obtained with respect to the people receiving the content with normal spreading.*

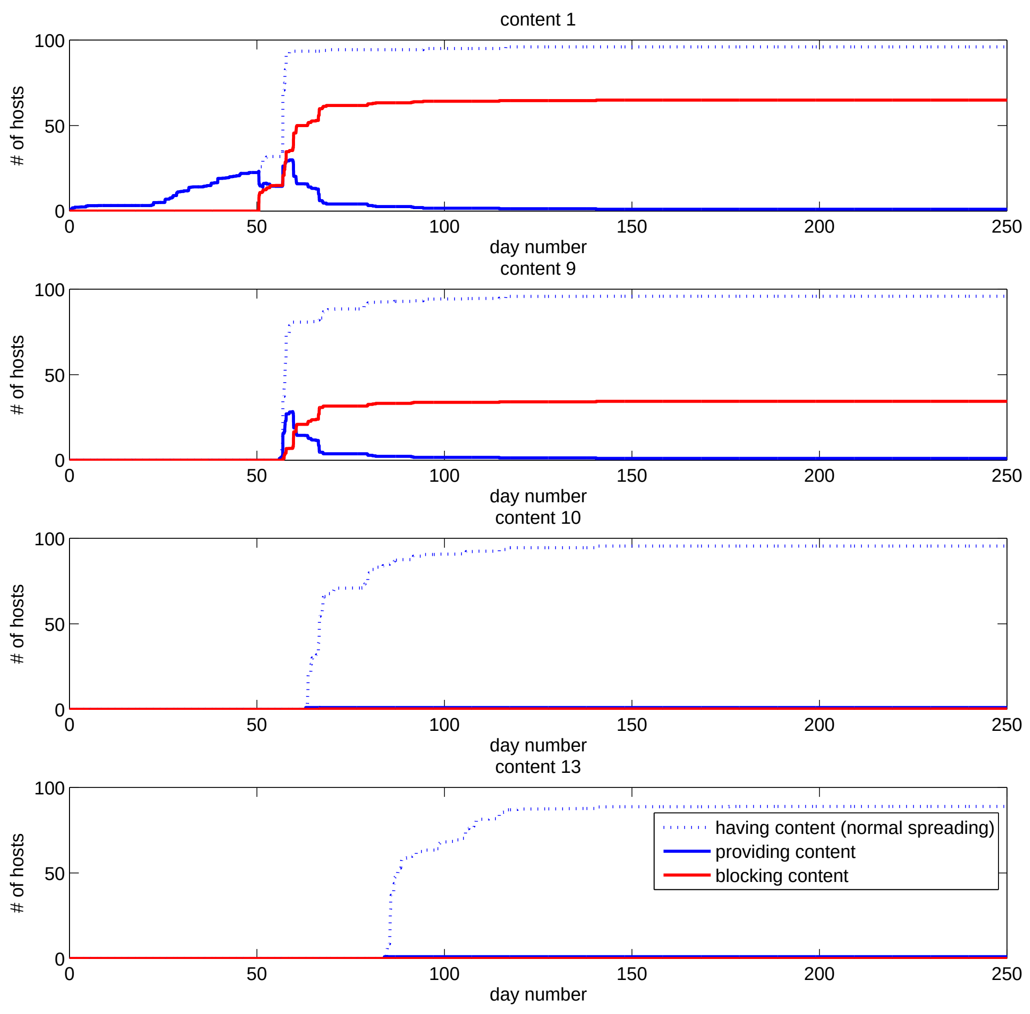


Figure C.7.: Moderator Blacklisting in Unclassified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. One moderator detects spam upon reception with recognition probability of 100% and exchanges that information with others which immediately accept the blacklist information. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of contents 1, 9, 10, 13 generated in the corresponding weeks. Huge increase at semester start at around day 60 because of much more connections.*

C.2.1. Classified MIT

classification	time	tot. having	max. providing (of all 96)
good	1.5h	79.41%	36.54%
average	14.8h	64.57%	18.99%
bad	36.5h	77.98%	24.41%

Table C.6.: Moderator Blacklists: *Time in hours when the maximum of users having the content is reached in good, average and bad classified MIT weeks.*

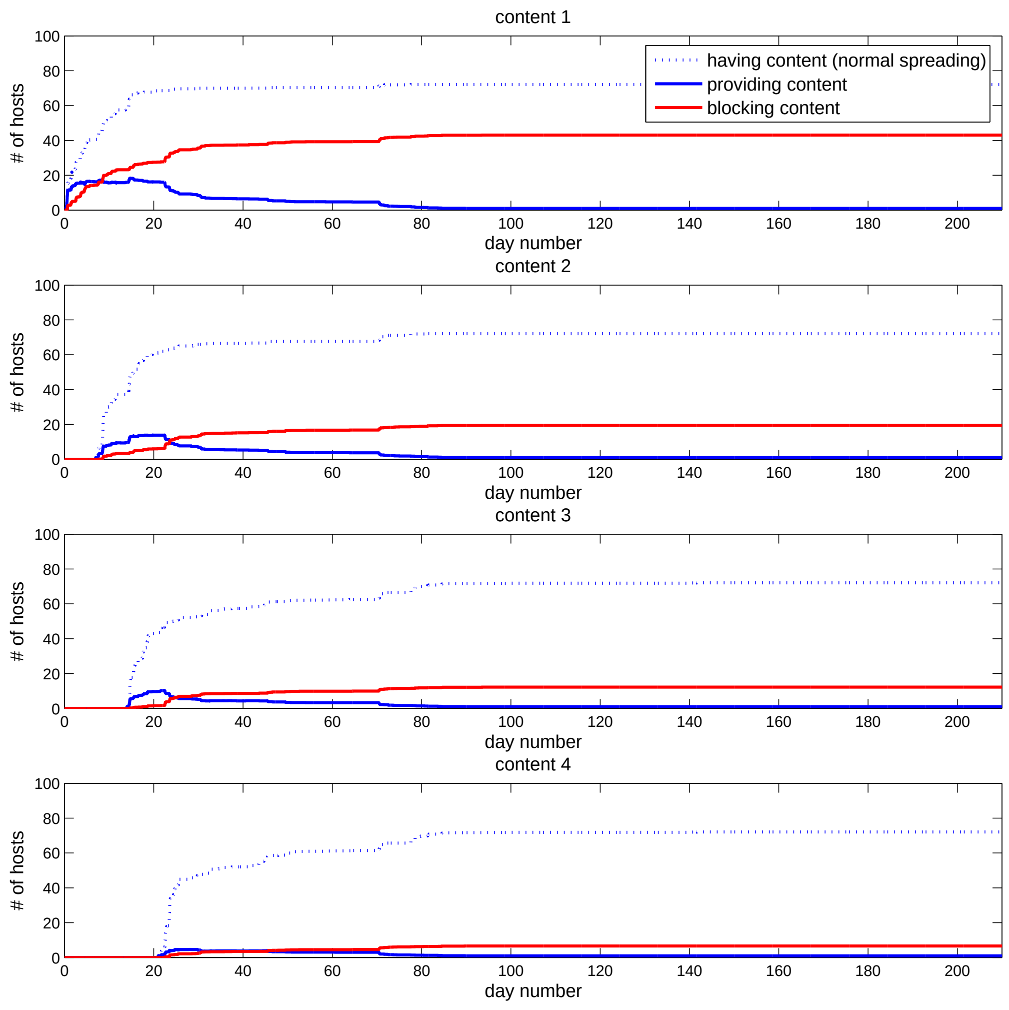


Figure C.8.: Moderator Blacklisting in Average Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. One moderator detects spam upon reception with recognition probability of 100% and exchanges that information with others which immediately accept the blacklist information. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

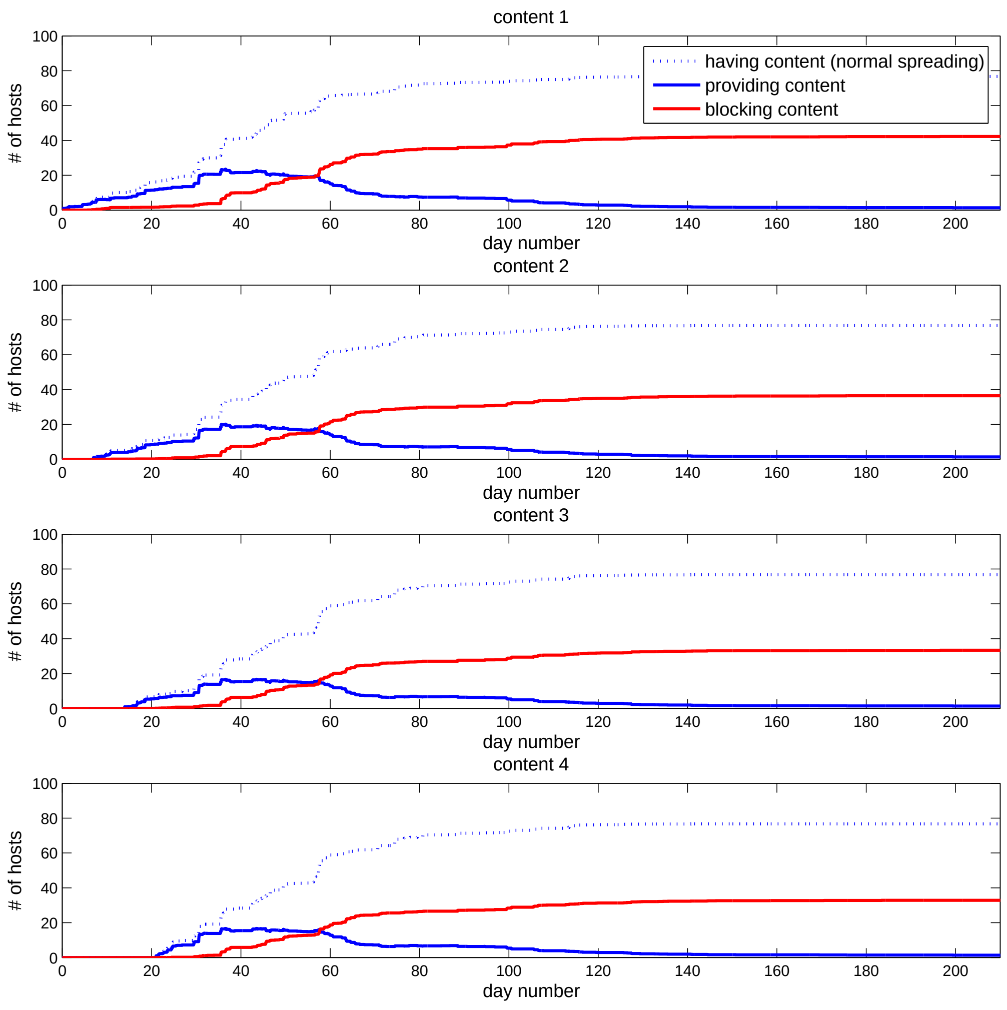


Figure C.9.: Moderator Blacklisting in Bad Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. One moderator detects spam upon reception with recognition probability of 100% and exchanges that information with others which immediately accept the blacklist information. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

C.2.2. Haggle Set

The moderator detects the malicious content already after 15 minutes. The maximum of people providing the content is achieved after 16 minutes and shown in Table C.10. At maximum 32.76% of all 41 people provide the content.

time	having	providing
15min	87.89%	67.16%
20h	4.84%	75.77%

Table C.7.: Moderator Blacklists: *Time in hours when the maximum of users having the content is reached in the Haggle data set. The percentages are relative to the normal spreading of the content.*

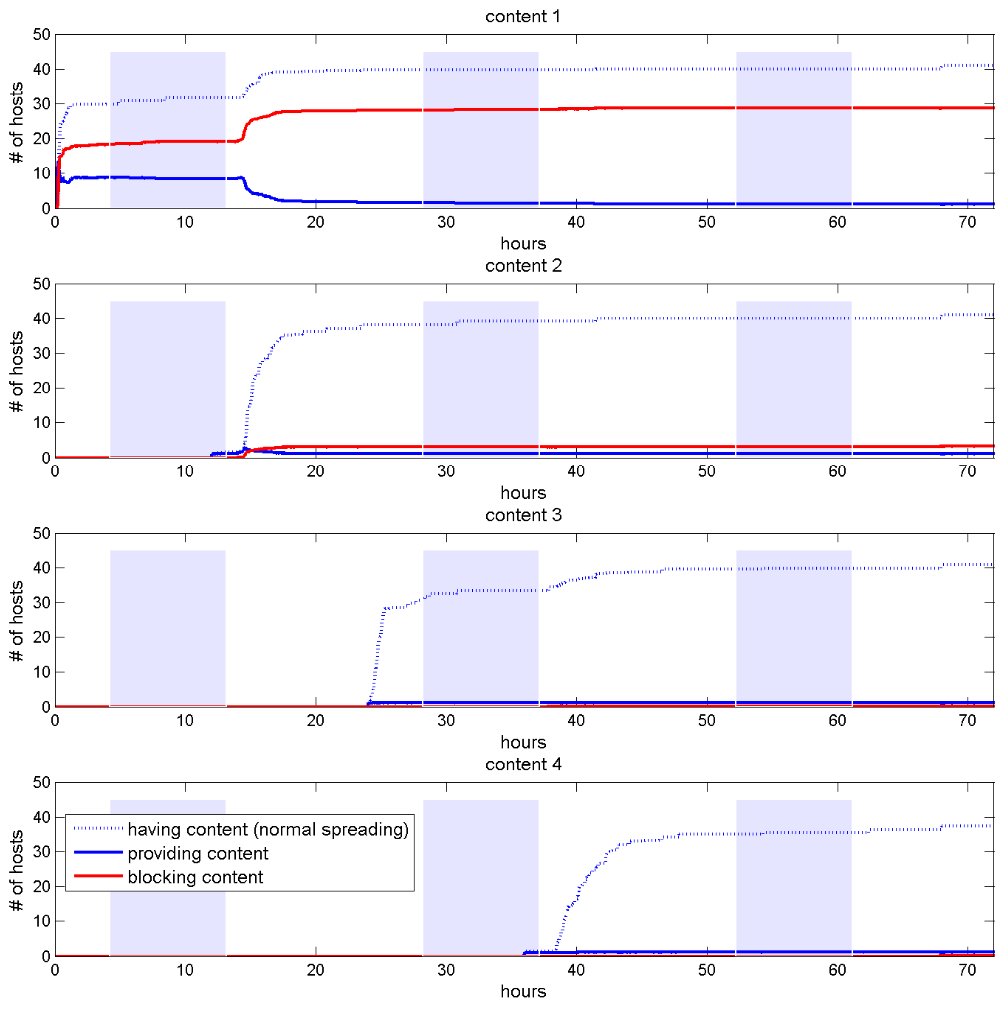


Figure C.10.: Moderator Blacklisting in Haggle Data Set: *Average spreading performance of one content generated by a spammer every 12 hours. One moderator detects spam upon reception with recognition probability of 100% and exchanges that information with others which immediately accept the blacklist information. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The gray shaded regions show an 8 hour inactivivity period during night. The graph presents the distribution of the first four contents generated in the first 36 hours.*

C.3. Personal Blacklisting

C.3.1. Unclassified MIT

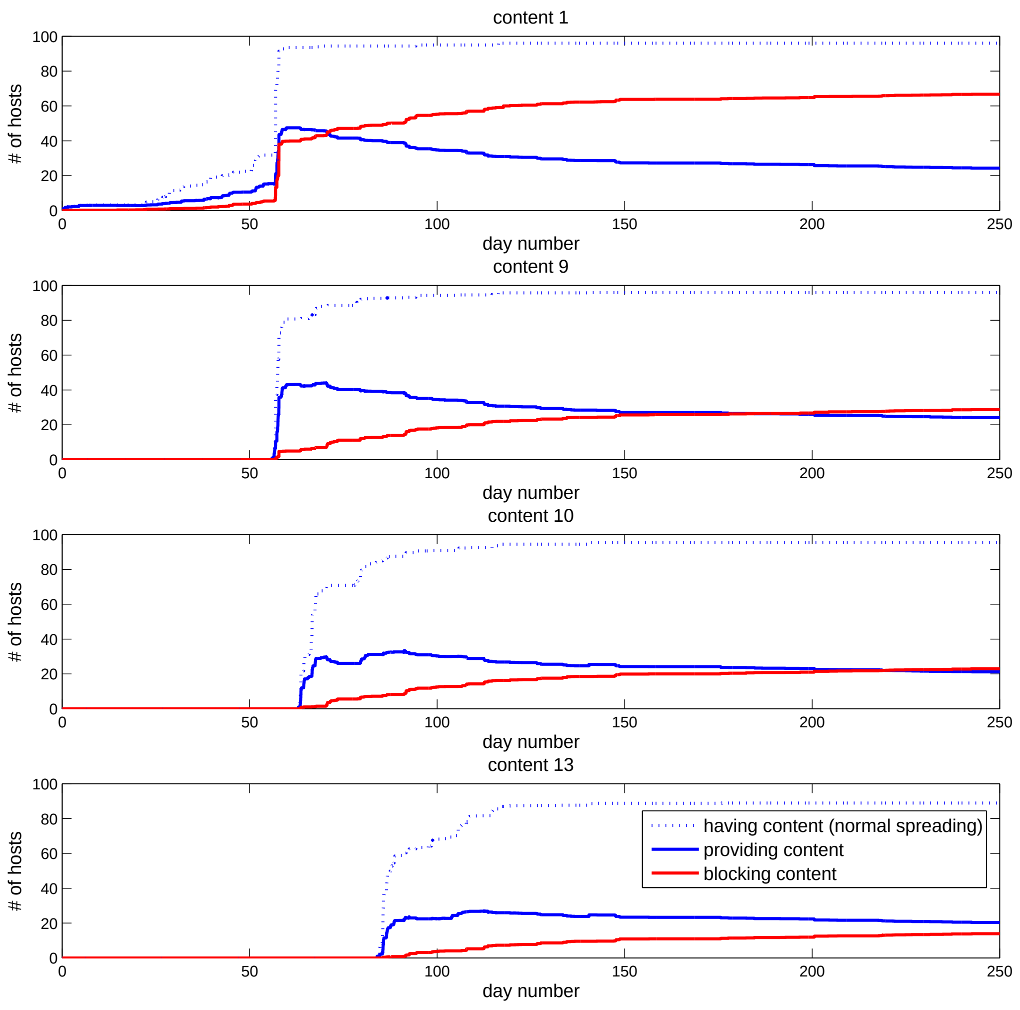


Figure C.11.: Personal Blacklisting in Unclassified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All the users keep that information secret not notifying others. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of contents 1, 9, 10, 13 generated in the corresponding weeks. Huge increase at semester start at around day 60 because of much more connections.*

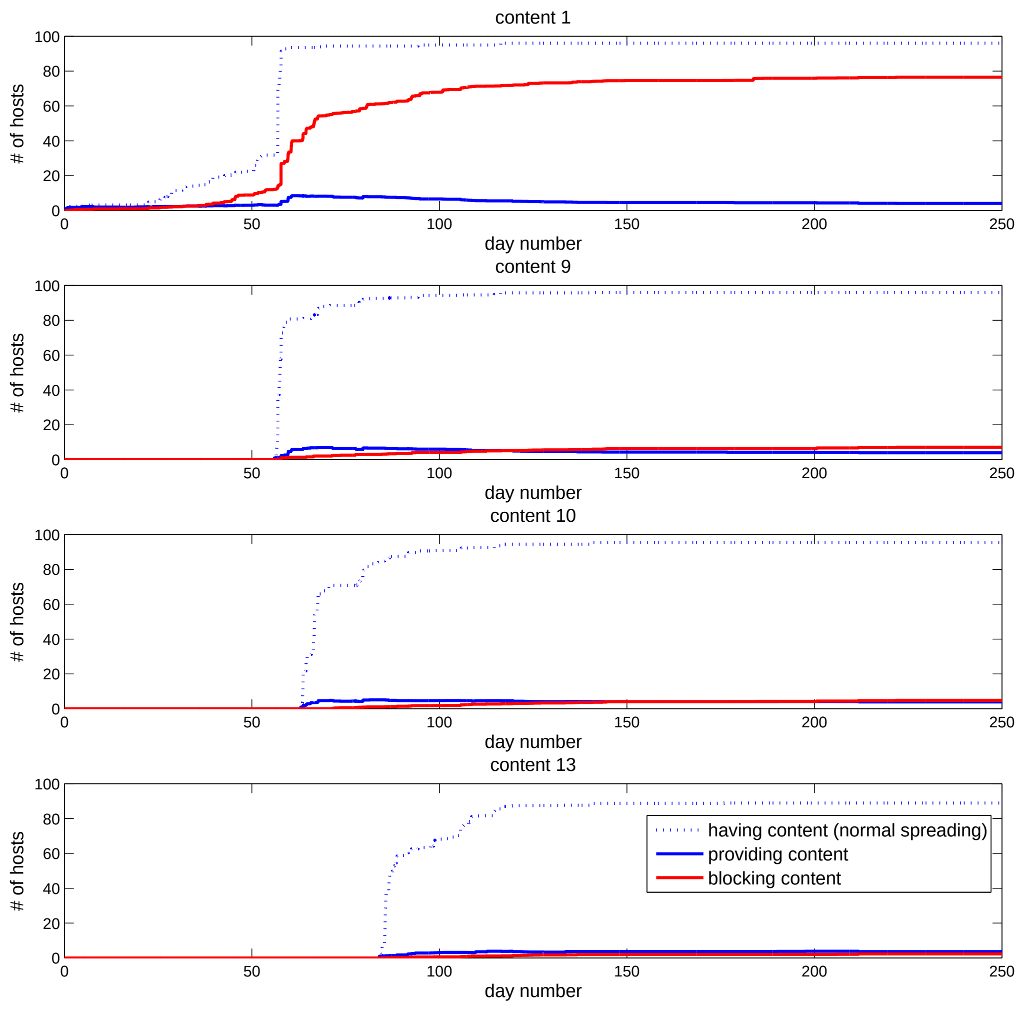


Figure C.12.: Personal Blacklisting in Unclassified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 50%. All the users keep that information secret not notifying others. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of contents 1, 9, 10, 13 generated in the corresponding weeks. Huge increase at semester start at around day 60 because of much more connections.*

C.3.2. Classified MIT

The maximum percentage of people providing the content can be found in Table C.8. The time, when the maximum point is reached increases for a decrease in the environment quality.

classification	time	tot. having content	max. providing (of all 96)
good	13.8d	99.90%	77.94%
average	16.5d	99.94%	59.19%
bad	60.0d	92.00%	41.44%

Table C.8.: Personal Blacklists: *Time in hours when the maximum of users having the content is reached in good, average and bad classified MIT weeks. The recognition probability is set to 10%.*

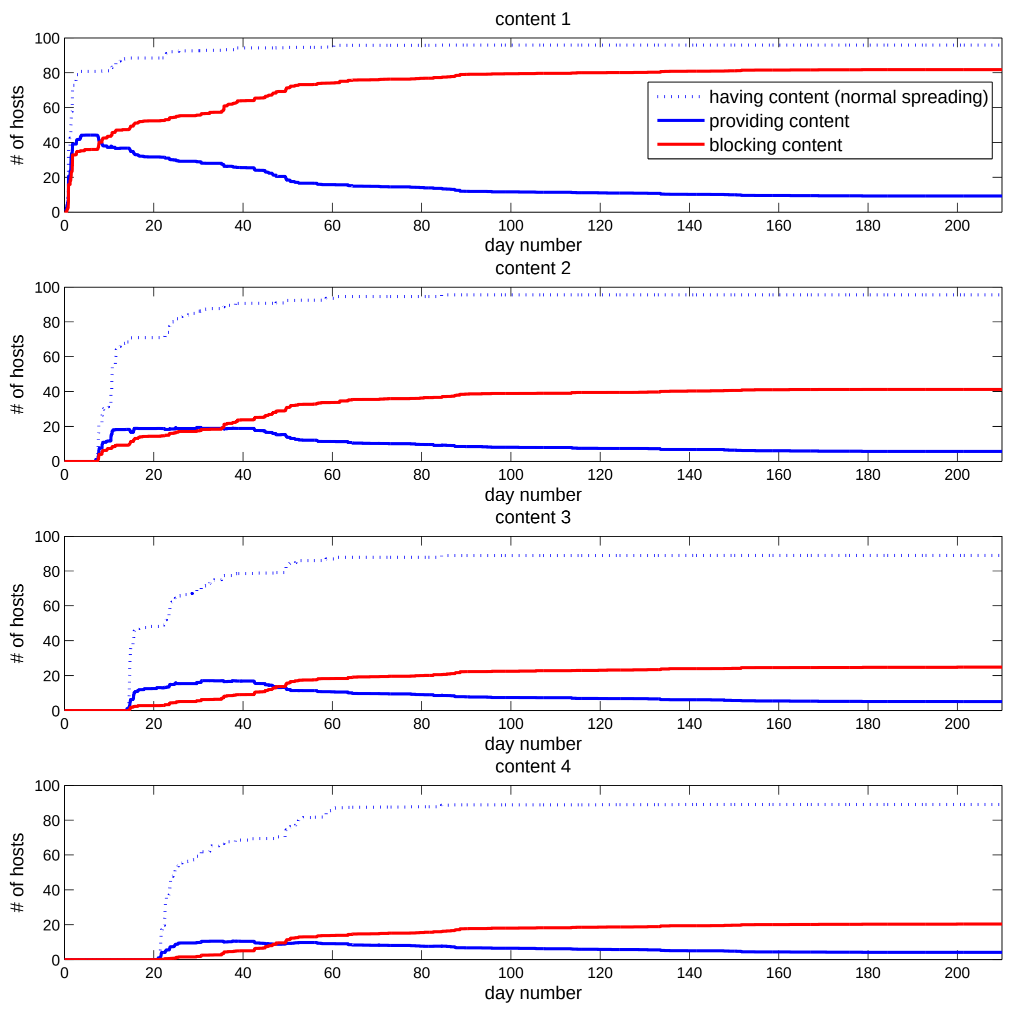


Figure C.13.: Personal Blacklisting in Good Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 50%. All the users keep that information secret not notifying others. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

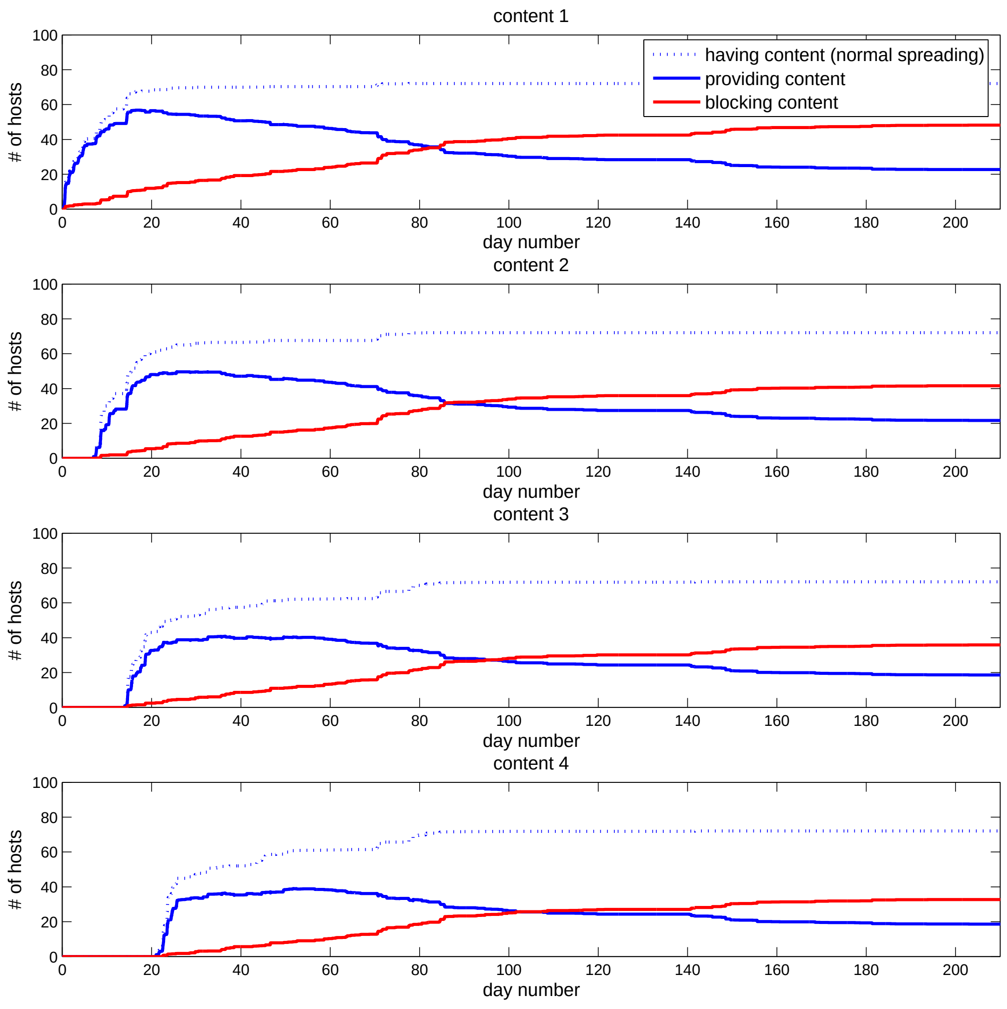


Figure C.14.: Personal Blacklisting in Average Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All the users keep that information secret not notifying others. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

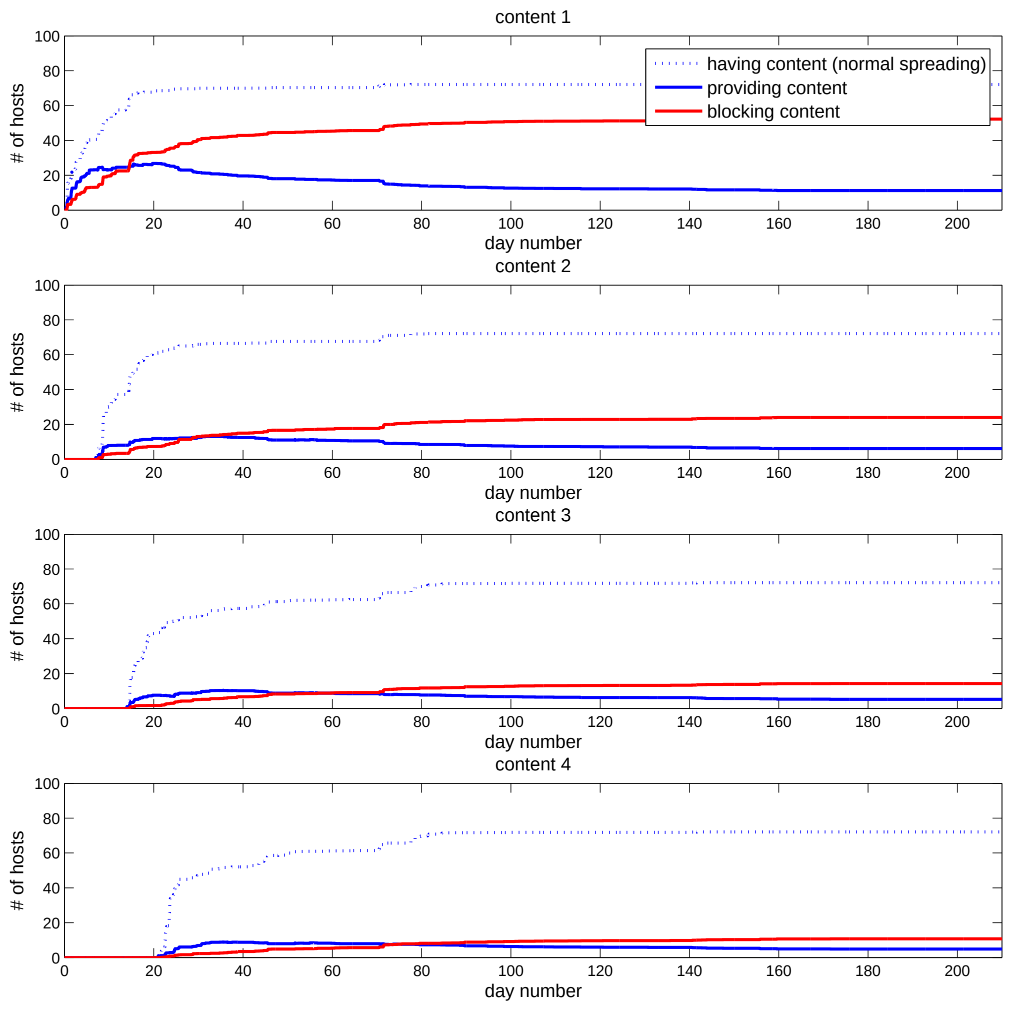


Figure C.15.: Personal Blacklisting in Average Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 50%. All the users keep that information secret not notifying others. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

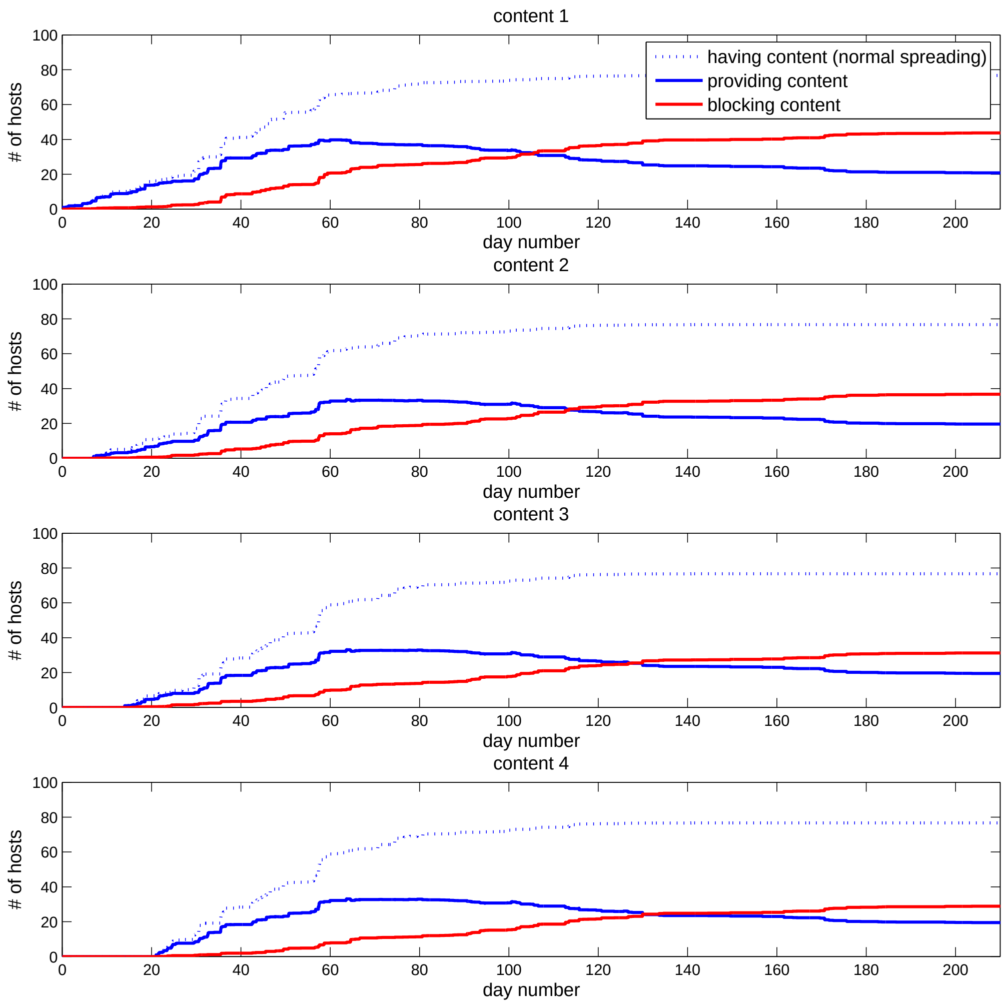


Figure C.16.: Personal Blacklisting in Bad Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All the users keep that information secret not notifying others. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

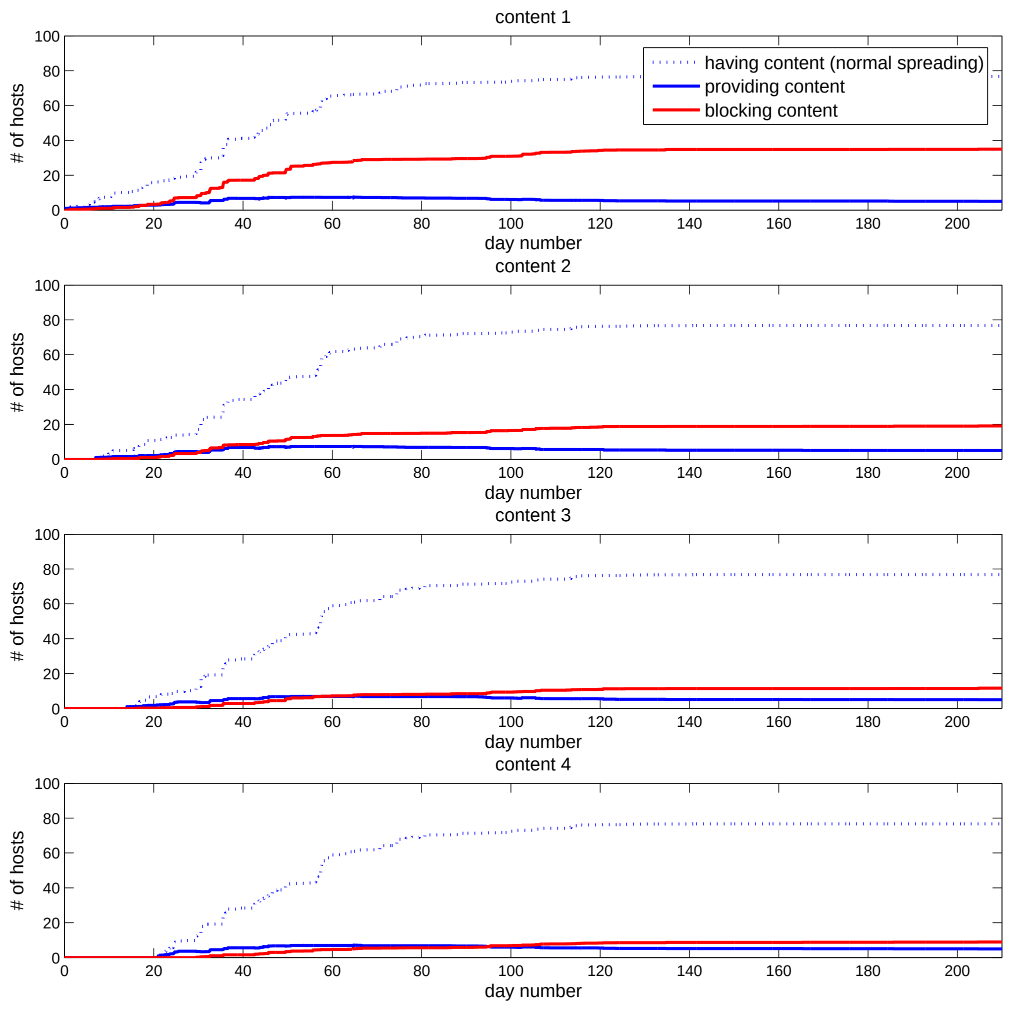


Figure C.17.: Personal Blacklisting in Bad Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 50%. All the users keep that information secret not notifying others. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

C.4. Local Blacklisting

C.4.1. Unclassified MIT

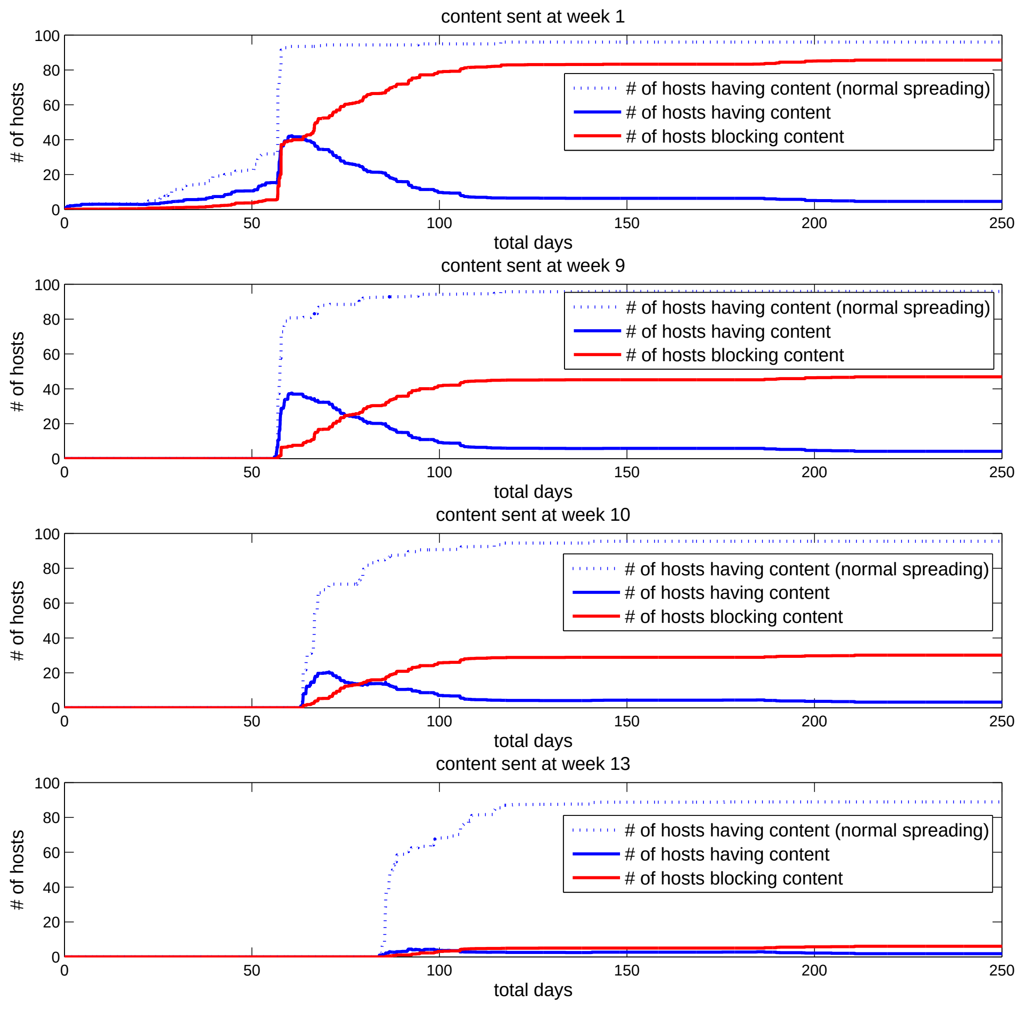


Figure C.18.: Local Blacklisting in Unclassified MIT Set: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 20 opinions. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of contents 1, 9, 10, 13 generated in the corresponding weeks. Huge increase at semester start at around day 60 because of much more connections.*

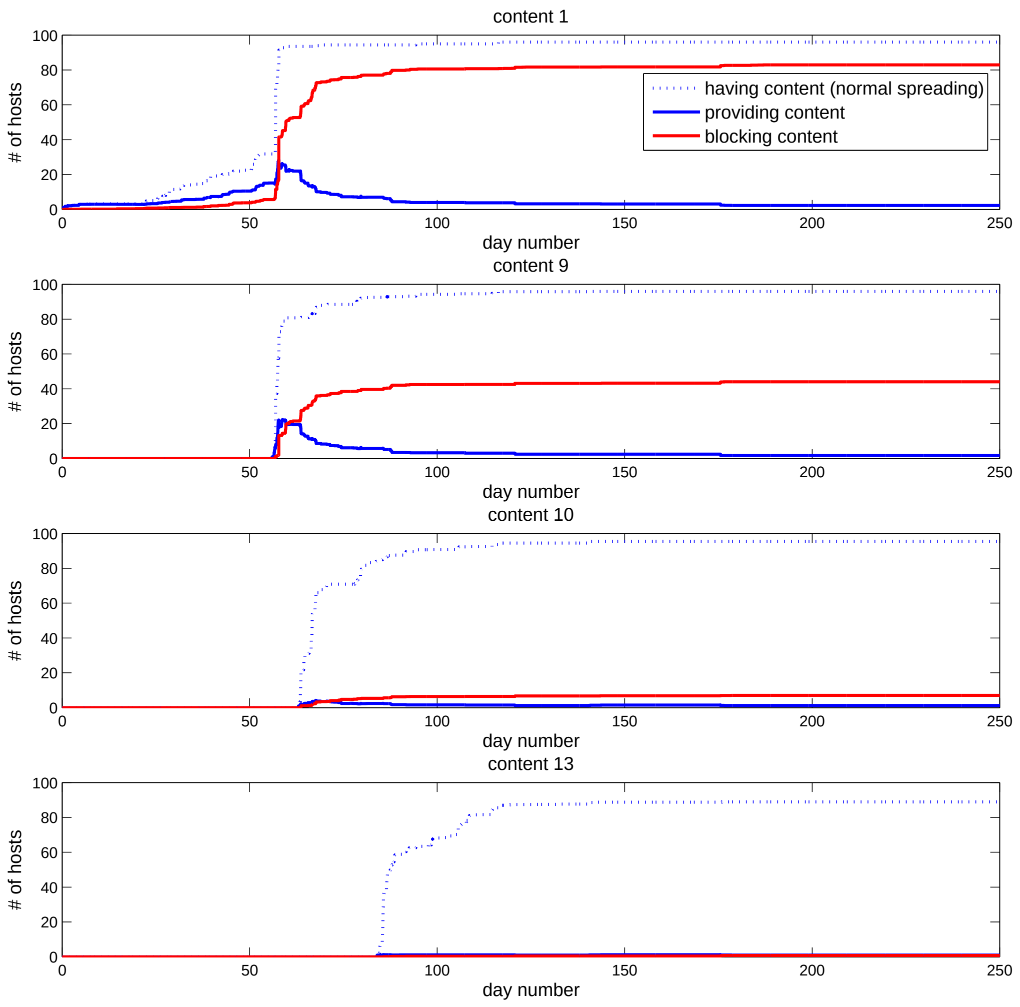


Figure C.19.: Local Blacklisting in Unclassified MIT Set: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 10 opinions. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of contents 1, 9, 10, 13 generated in the corresponding weeks. Huge increase at semester start at around day 60 because of much more connections.*

C.4.2. Classified MIT

The maximum of people providing content when using local blacklist with a suggestion threshold of 20 recommendations and a recognition probability of 10% can be found in Table C.9.

classification	time	total having content	max providing (of all 96)
good	13.8d	99.90%	77.94%
average	16.5d	99.94%	59.19%
bad	60.0d	92.00%	41.44%

Table C.9.: Local Blacklists: *Time in hours when the maximum of users having the content is reached in good, average and bad classified MIT weeks. The recognition probability is set to 10% and the suggestion threshold equals 20 opinions.*

	Good		Average		Bad	
time	having	providing	having	providing	having	providing
1 week	100%	90.36%	100%	92.60%	100%	94.74%
2 weeks	99.90%	84.07%	97.02%	84.34%	95.28%	87.98%
4 weeks	96.52%	59.87%	100%	74.89%	95.96%	83.41%
8 weeks	95.95%	13.52%	99.57%	53.31%	89.07%	50.58%
16 weeks	95.83%	9.71%	98.25%	29.33%	77.03%	22.74%

Table C.10.: Local Blacklists: *The table lists the percentages of people providing and having the first content in good, average and bad classified MIT weeks. The week times are relative to day 0 and the percentages to the people having the content with normal spreading. The spam recognition is set to 10% and the suggestion threshold equals 10 opinions.*

classification	time	total having content	max providing (of all 96)
good	12.5d	99.95%	77.63%
average	16.5d	99.94%	59.10%
bad	46.5d	91.59%	31.97%

Table C.11.: Local Blacklists: *Time in hours when the maximum of users having the content is reached in good, average and bad classified MIT weeks. The recognition probability is set to 10% and the suggestion threshold equals 10 opinions.*

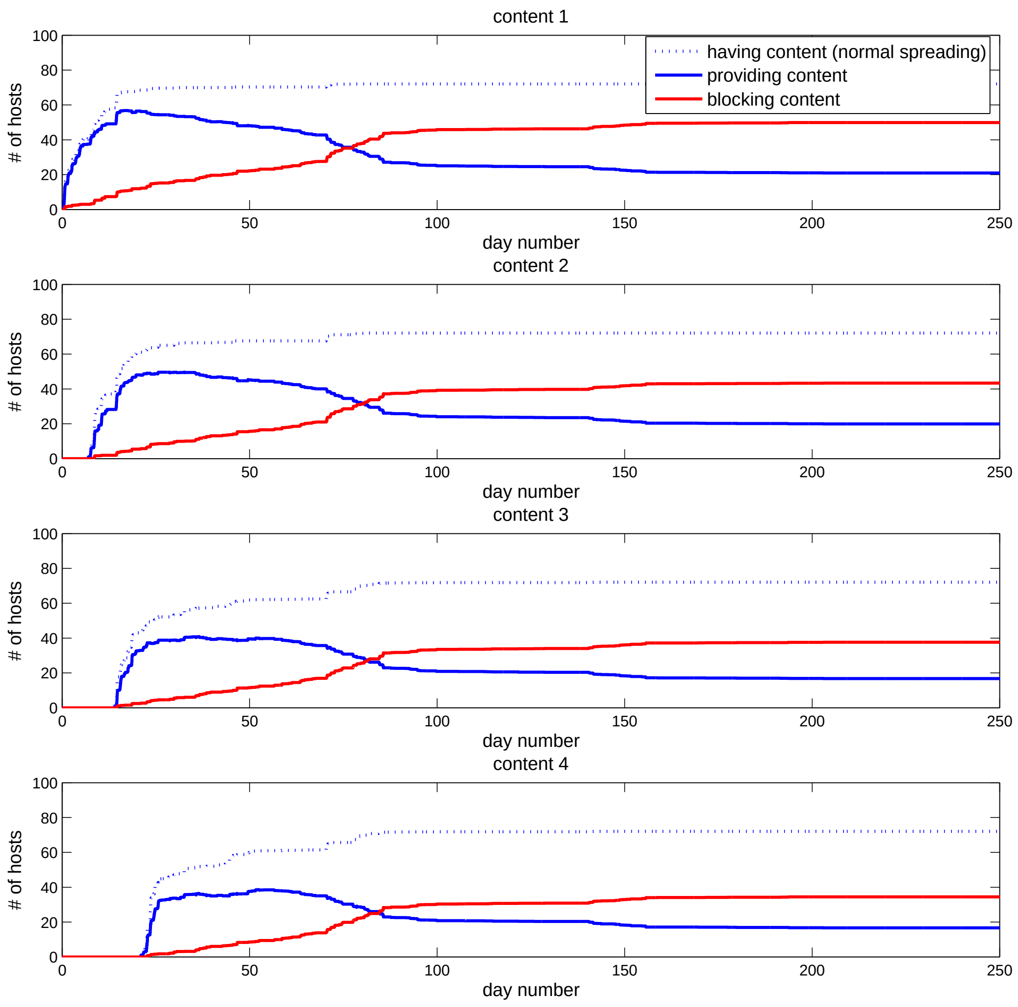


Figure C.20.: Local Blacklisting in Average Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 20 opinions. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

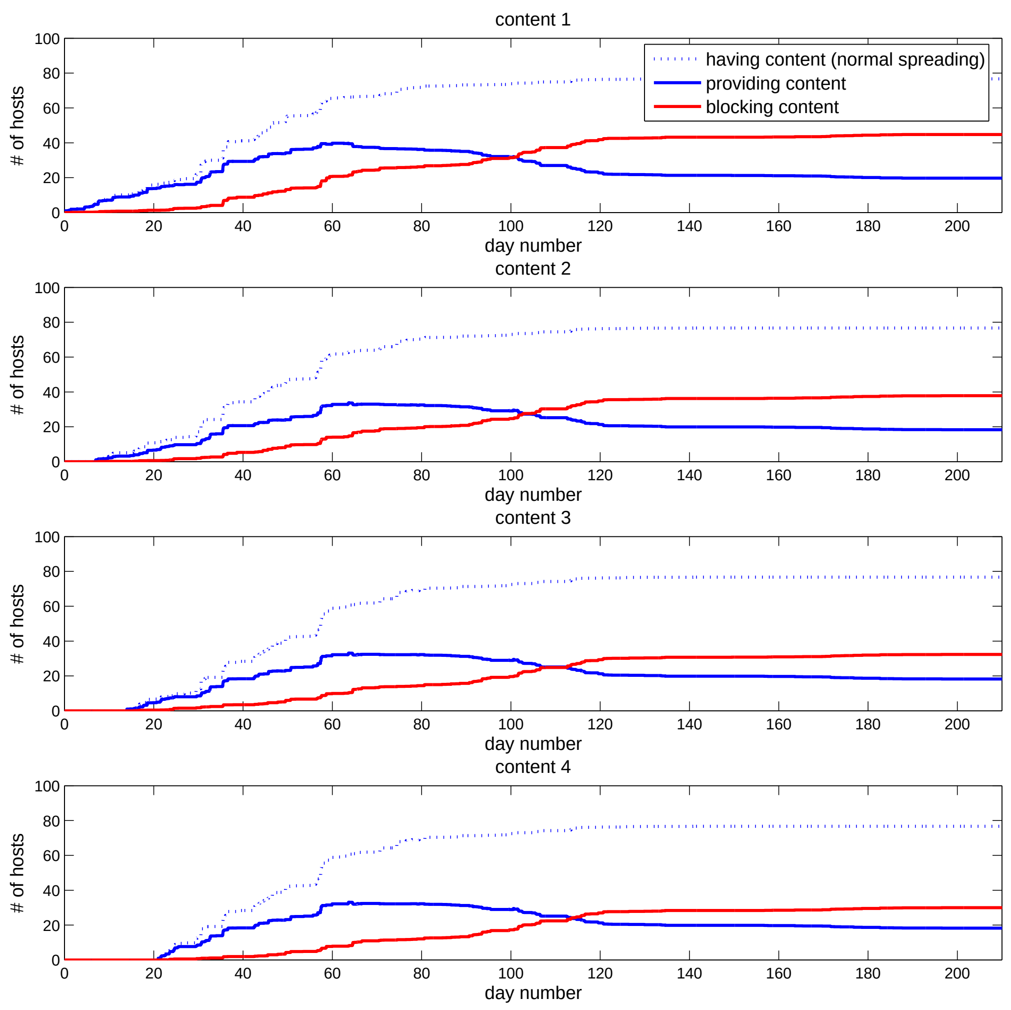


Figure C.21.: Local Blacklisting in Bad Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 20 opinions. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

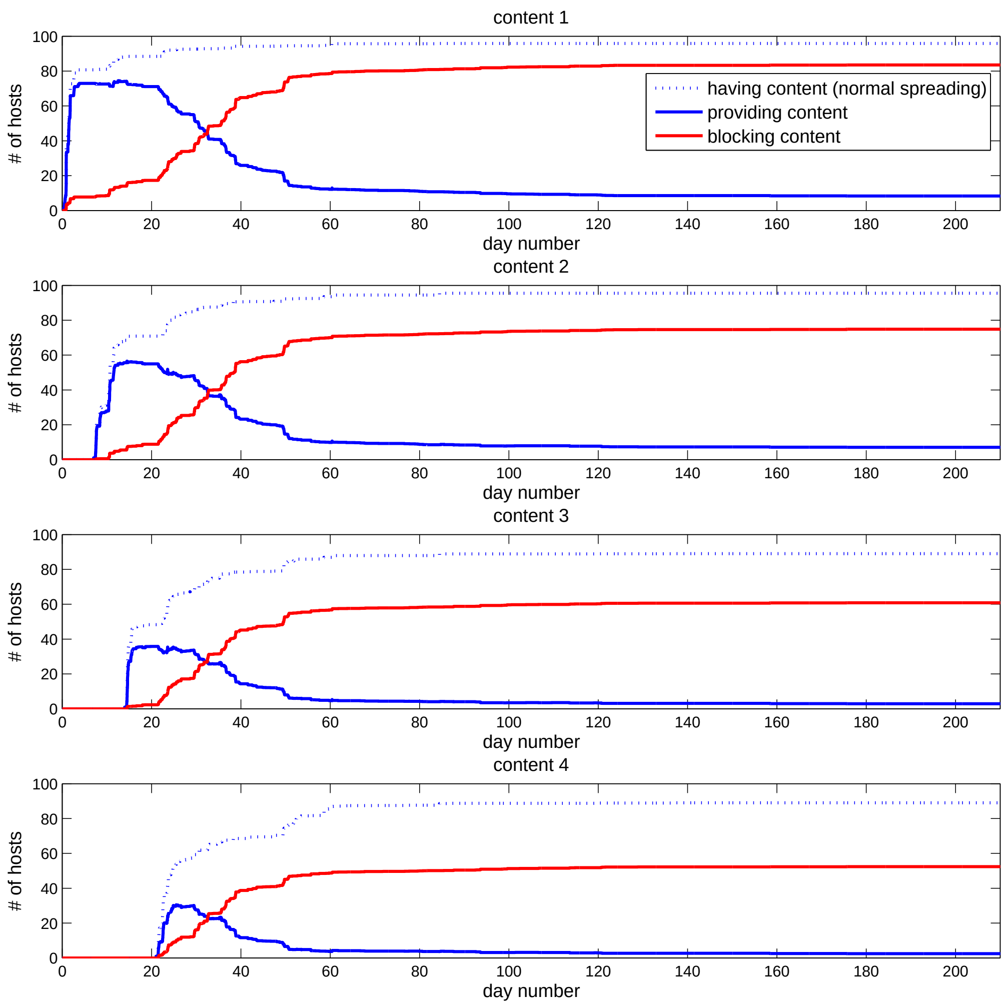


Figure C.22.: Local Blacklisting in Good Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 10 opinions. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

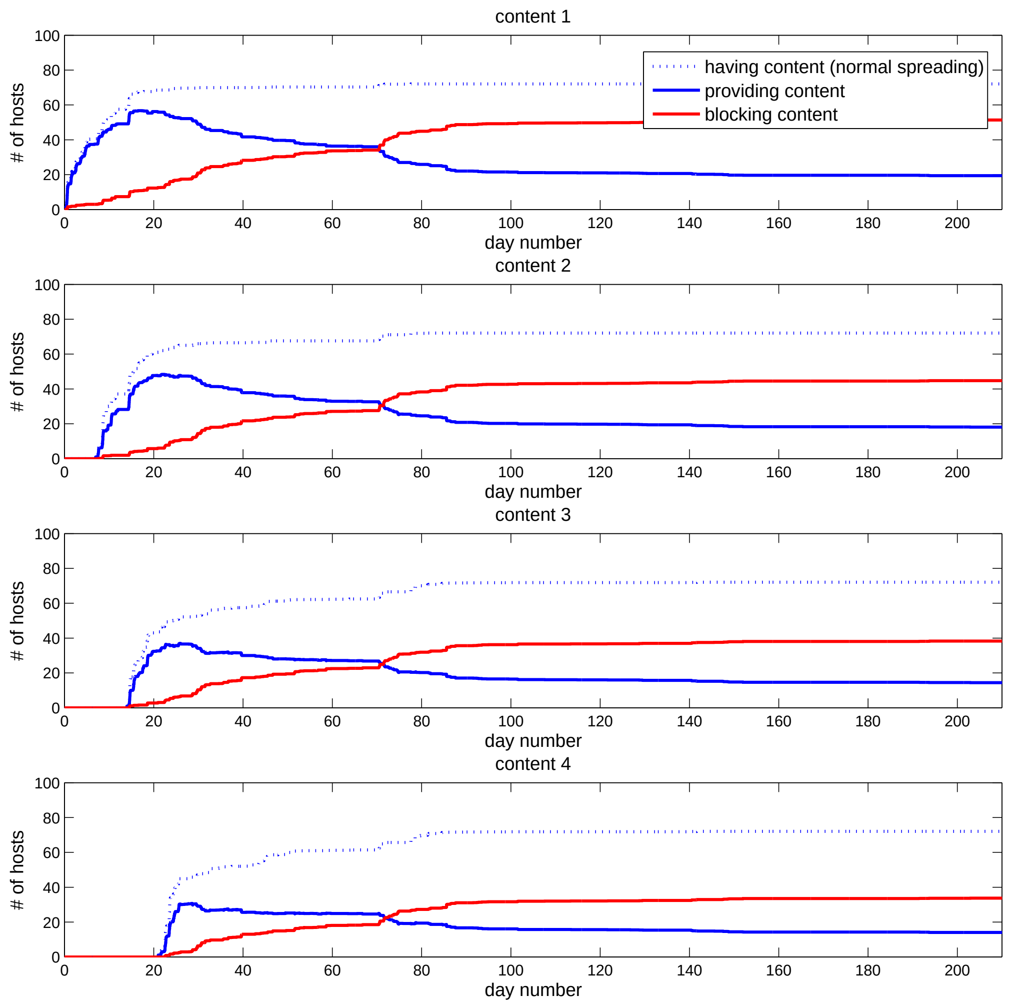


Figure C.23.: Local Blacklisting in Average Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 10 opinions. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

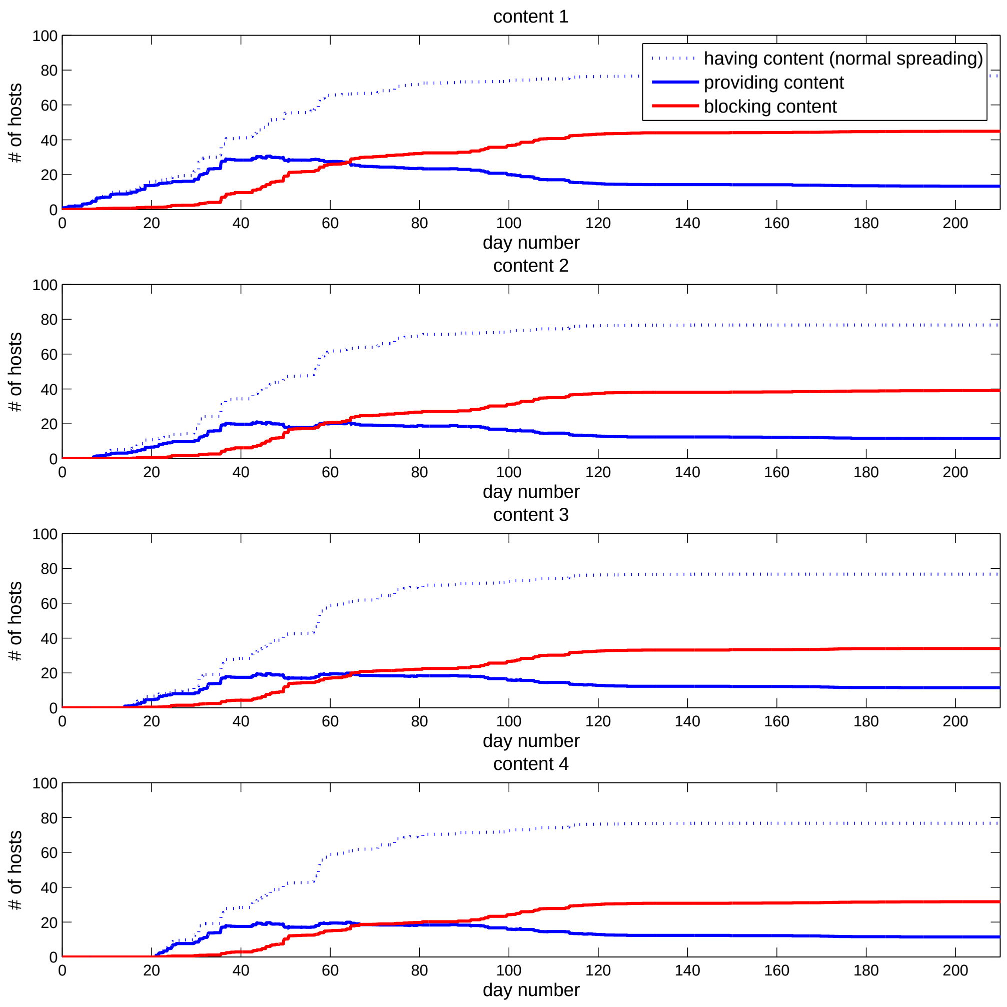


Figure C.24.: Local Blacklisting in Bad Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every week. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 10 opinions. The red and blue line together show the number of people that have received the content. The blue line presents the number of people that provide the received content to others whereas the red line show the persons that block the content after reception. The graph shows the distribution of the first four contents generated in the first four weeks.*

C.4.3. Synthetic Traces

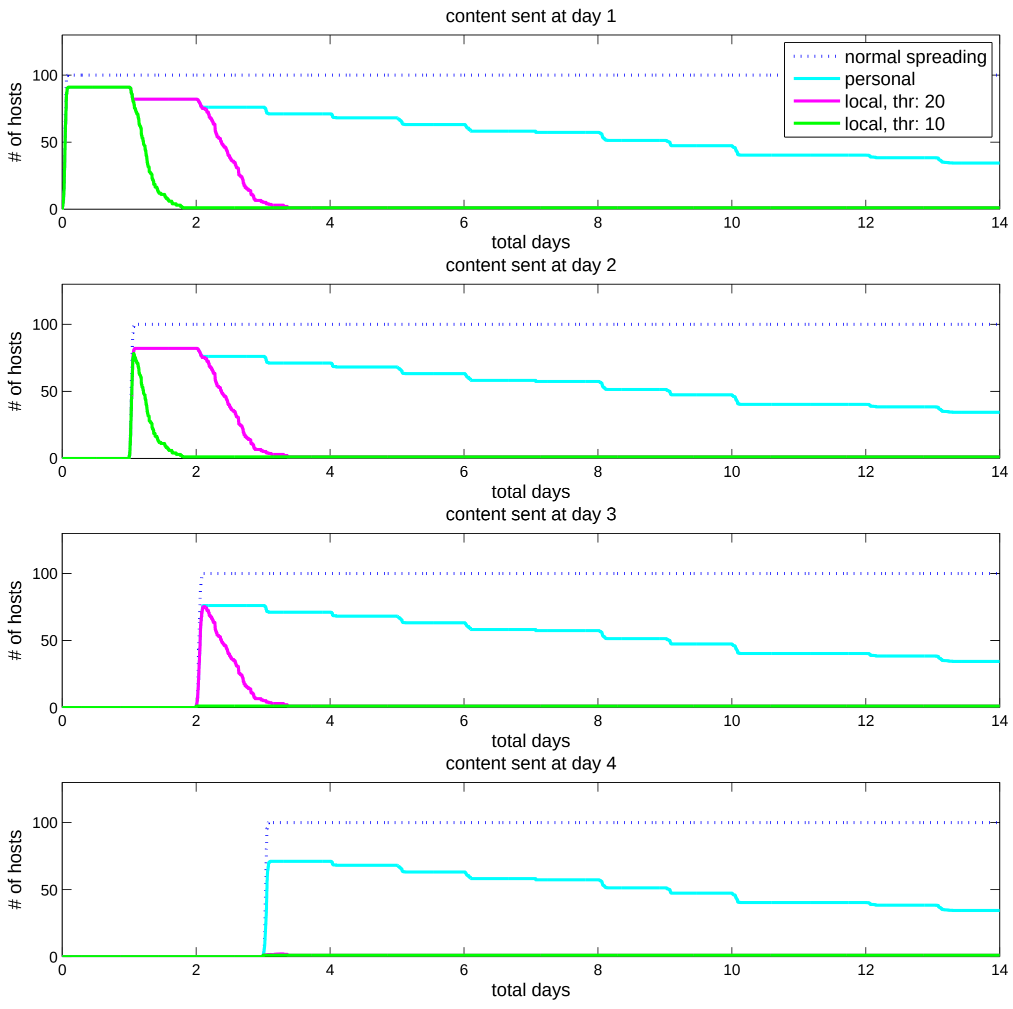


Figure C.25.: Local Blacklisting in Random Waypoint Model: *Average spreading performance of one content generated by a spammer every day. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 20 and 10 opinions. The colored lines present the number of people providing the corresponding content. The graph shows the distribution of the first four contents generated in the first four days. The detailed model parameters can be found in Section 4.2.*

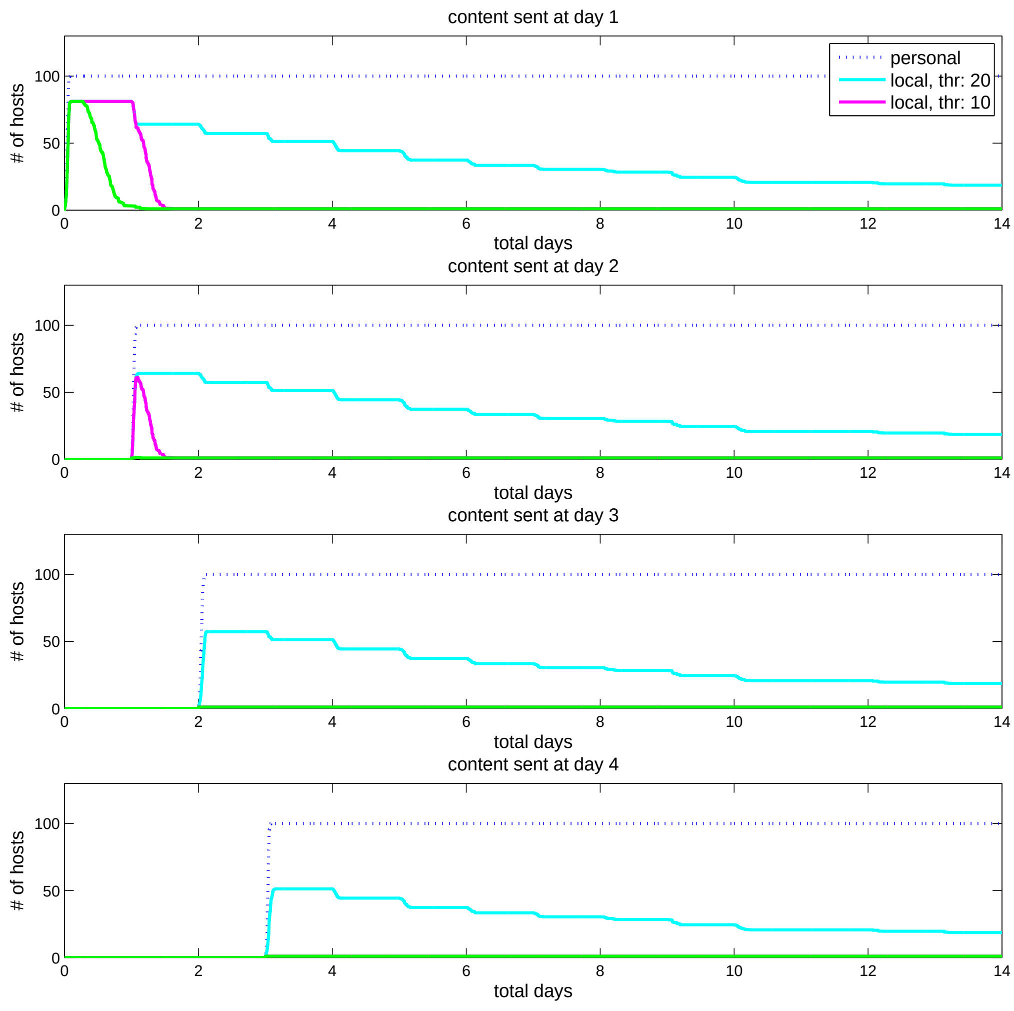


Figure C.26.: Local Blacklisting in Random Waypoint Model: *Average spreading performance of one content generated by a spammer every day. Every user detects malicious content upon reception with a recognition probability of 20%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 20 and 10 opinions. The colored lines present the number of people providing the corresponding content. The graph shows the distribution of the first four contents generated in the first four days. The detailed model parameters can be found in Section 4.2.*

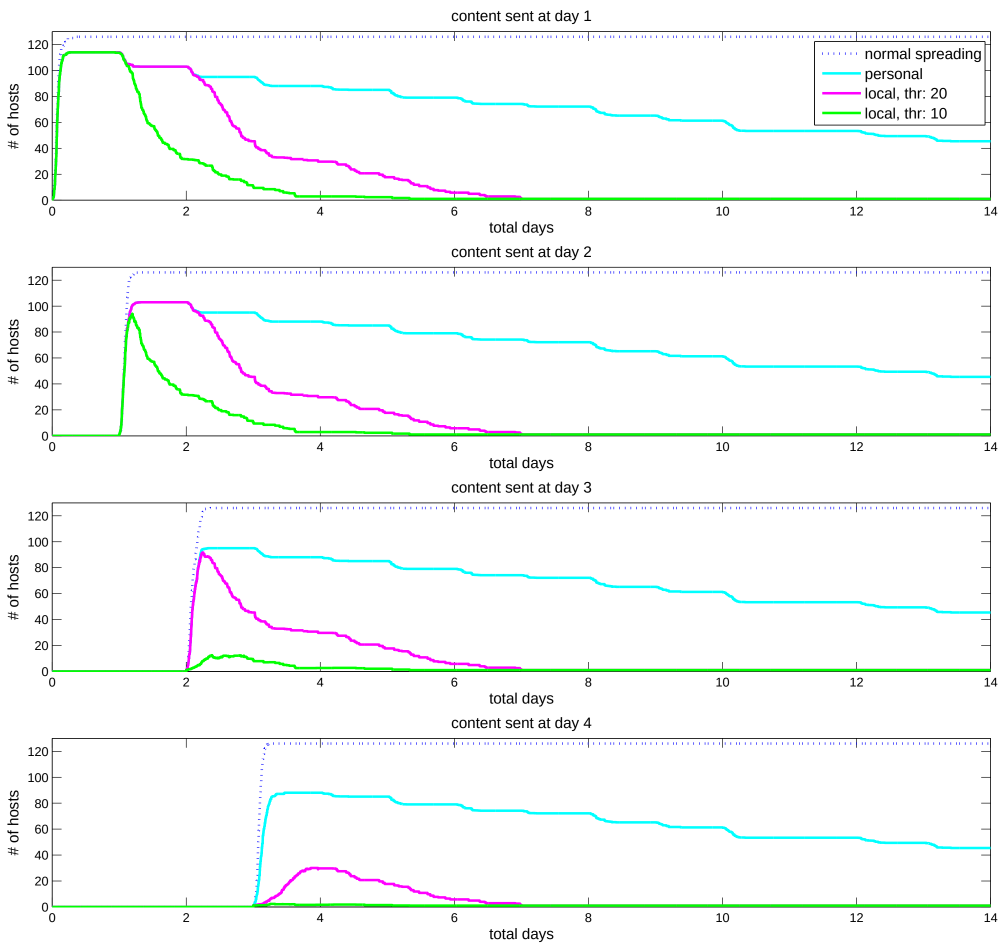


Figure C.27.: Local Blacklisting in Helsinki Model: *Average spreading performance of one content generated by a spammer every day. Every user detects malicious content upon reception with a recognition probability of 10%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 20 and 10 opinions. The colored lines present the number of people providing the corresponding content. The graph shows the distribution of the first four contents generated in the first four days. The parameter information can be found in Section 4.2.*

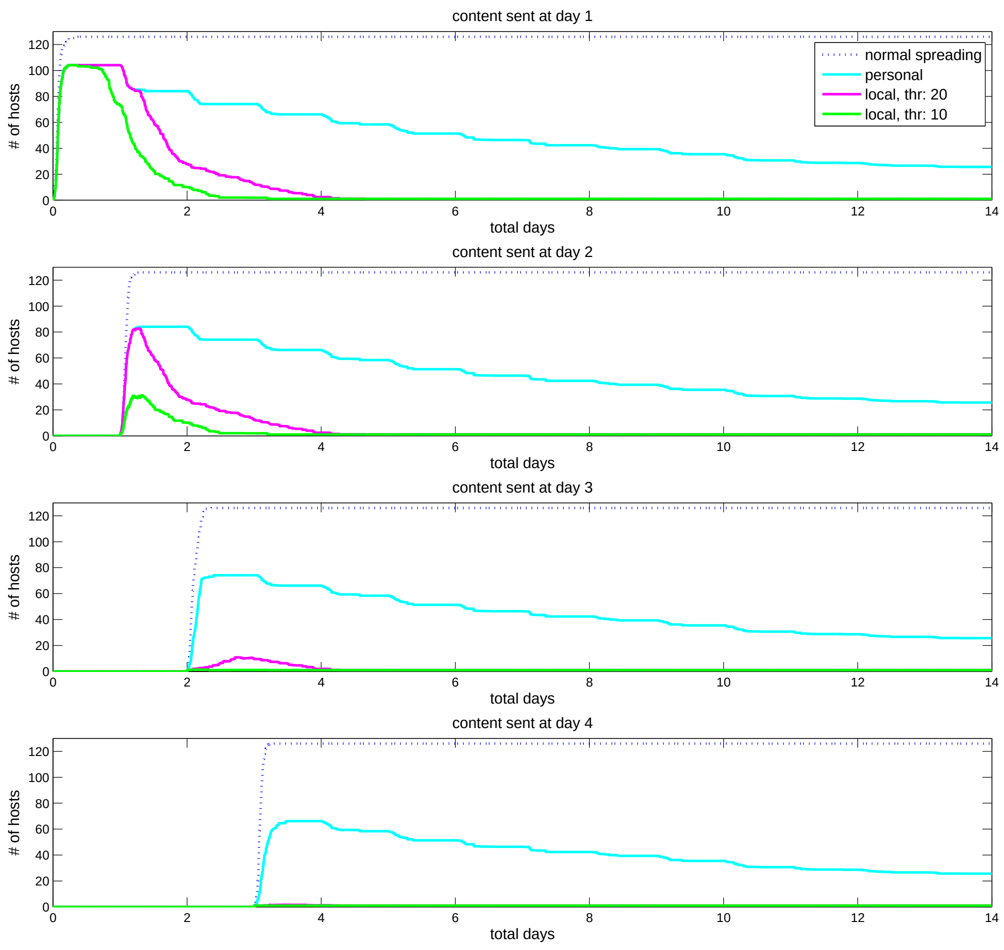


Figure C.28.: Local Blacklisting in Helsinki Model: *Average spreading performance of one content generated by a spammer every day. Every user detects malicious content upon reception with a recognition probability of 20%. All users exchange blacklist information with others and accept received information if it exceeds the suggestion threshold of 20 and 10 opinions. The colored lines present the number of people providing the corresponding content. The graph shows the distribution of the first four contents generated in the first four days. The parameter information can be found in Section 4.2.*

C.5. Send Rate Limitation

C.5.1. Classified MIT

	Good		Average		Bad	
time	having	providing	having	providing	having	providing
1 week	100%	86.70%	99.41%	84.98%	95.51%	85.90%
2 weeks	99.12%	77.44%	97.32%	77.03%	86.74%	75.99%
4 weeks	99.22%	55.89%	99.34%	67.39%	89.11%	79.23%
8 weeks	99.28%	18.40%	99.94%	53.12%	91.36%	72.29%
16 weeks	99.29%	14.11%	98.34%	36.26%	93.51%	60.54%

Table C.12.: Local Blacklists Combined with Send Rate Limitation: *The suggestion threshold is set to 20 opinions and the recognition probability equals 10%. The table lists the percentages of people providing and having the first content in good, average and bad classified MIT weeks. The week times are all relative to day 0 where the first content is generated. The percentages are obtained with respect to the people receiving the content with normal spreading.*

classification	time	total having content	max providing (of all 96)
good	3.8d	100%	75.82%
average	24.4d	99.26%	52.90%
bad	60.0d	95.59%	56.30%

Table C.13.: Local Blacklist Combined with Send Rate Limitation: *Time in hours when the maximum of users having the content is reached in good, average and bad classified MIT weeks. The spam recognition is set to 10% and the suggestion threshold equals 20 opinions.*

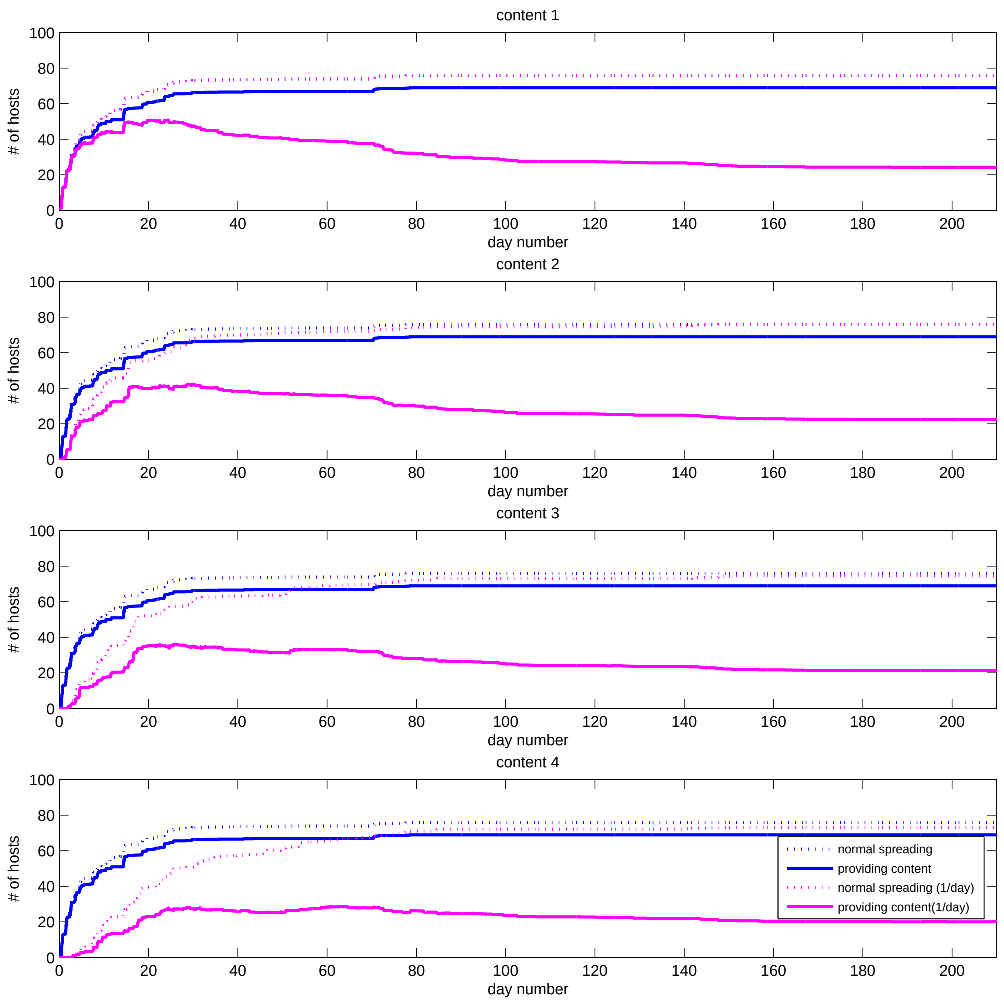


Figure C.29.: Effect of Send Rate in Average Classified MIT Weeks:*Average spreading performance of 300 contents generated by a spammer at day 0. Every user detects spam with a recognition probability of 10% and exchanges that information with others. Received information is accepted after exceeding a suggestion threshold of 20 opinions. Without send rate of 1 content/day, all 300 contents are rated only once at first reception. When using the send rate, every user rates received content every day after reception. The graph shows the people providing the first four contents of the burst.*

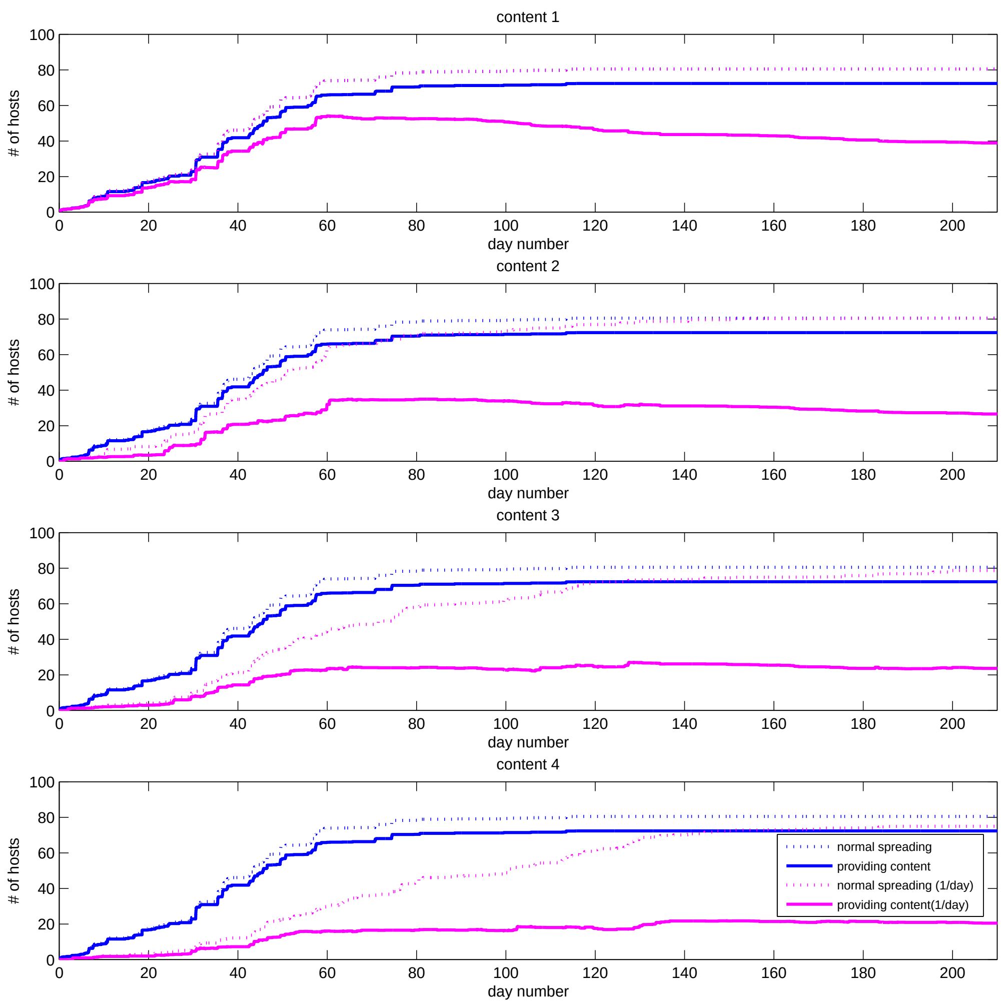


Figure C.30.: Effect of Send Rate in Bad Classified MIT Weeks:*Average spreading performance of 300 contents generated by a spammer at day 0. Every user detects spam with a recognition probability of 10% and exchanges that information with others. Received information is accepted after exceeding a suggestion threshold of 20 opinions. Without send rate of 1 content/day, all 300 contents are rated only once at first reception. When using the send rate, every user rates received content every day after reception. The graph shows the people providing the first four contents of the burst.*

	Good		Average		Bad	
time	having	providing	having	providing	having	providing
1 week	100%	85.96%	99.41%	84.98%	95.51%	85.90%
2 weeks	99.12%	68.21%	96.78%	72.21%	86.74%	75.99%
4 weeks	99.22%	48.64%	99.28%	57.52%	89.11%	79.23%
8 weeks	99.04%	11.50%	99.53%	46.17%	91.16%	65.65%
16 weeks	98.72%	8.52%	98.16%	30.98%	92.92%	54.11%

Table C.14.: Local Blacklists Combined with Send Rate Limitation: *The suggestion threshold is set to 10 opinions and the recognition probability equals 10%. The table lists the percentages of people providing and having the first content in good, average and bad classified MIT weeks. The week times are all relative to day 0 where the first content is generated. The percentages are obtained with respect to the people receiving the content with normal spreading.*

classification	time	total having content	max providing (of all 96)
good	3.8d	100%	75.41%
average	24.4d	99.79%	48.23%
bad	60.0d	94.82%	52.76%

Table C.15.: Local Blacklist Combined with Send Rate Limitation: *Time in hours when the maximum of users having the content is reached in good, average and bad classified MIT weeks. The spam recognition is set to 10% and the suggestion threshold equals 10 opinions.*

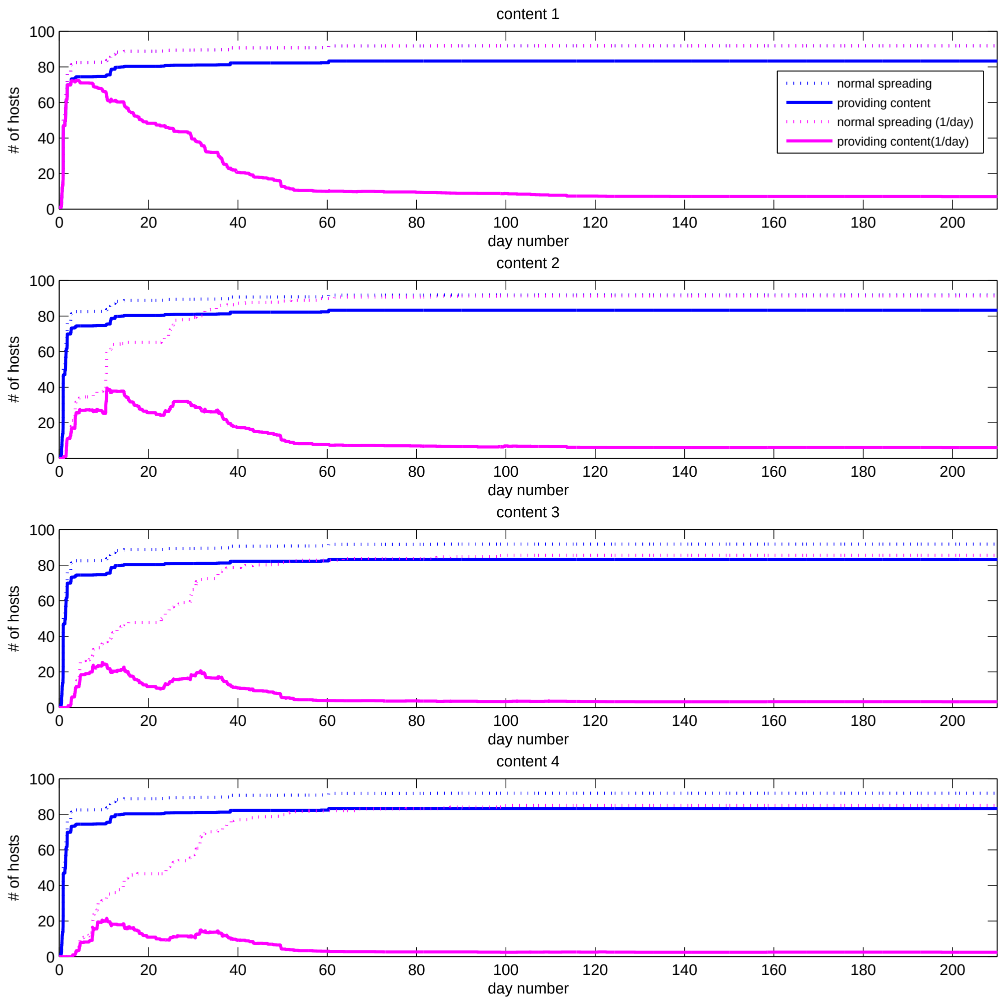


Figure C.31.: Effect of Send Rate in Good Classified MIT Weeks:*Average spreading performance of 300 contents generated by a spammer at day 0. Every user detects spam with a recognition probability of 10% and exchanges that information with others. Received information is accepted after exceeding a suggestion threshold of 10 opinions. Without send rate of 1 content/day, all 300 contents are rated only once at first reception. When using the send rate, every user rates received content every day after reception. The graph shows the people providing the first four contents of the burst.*

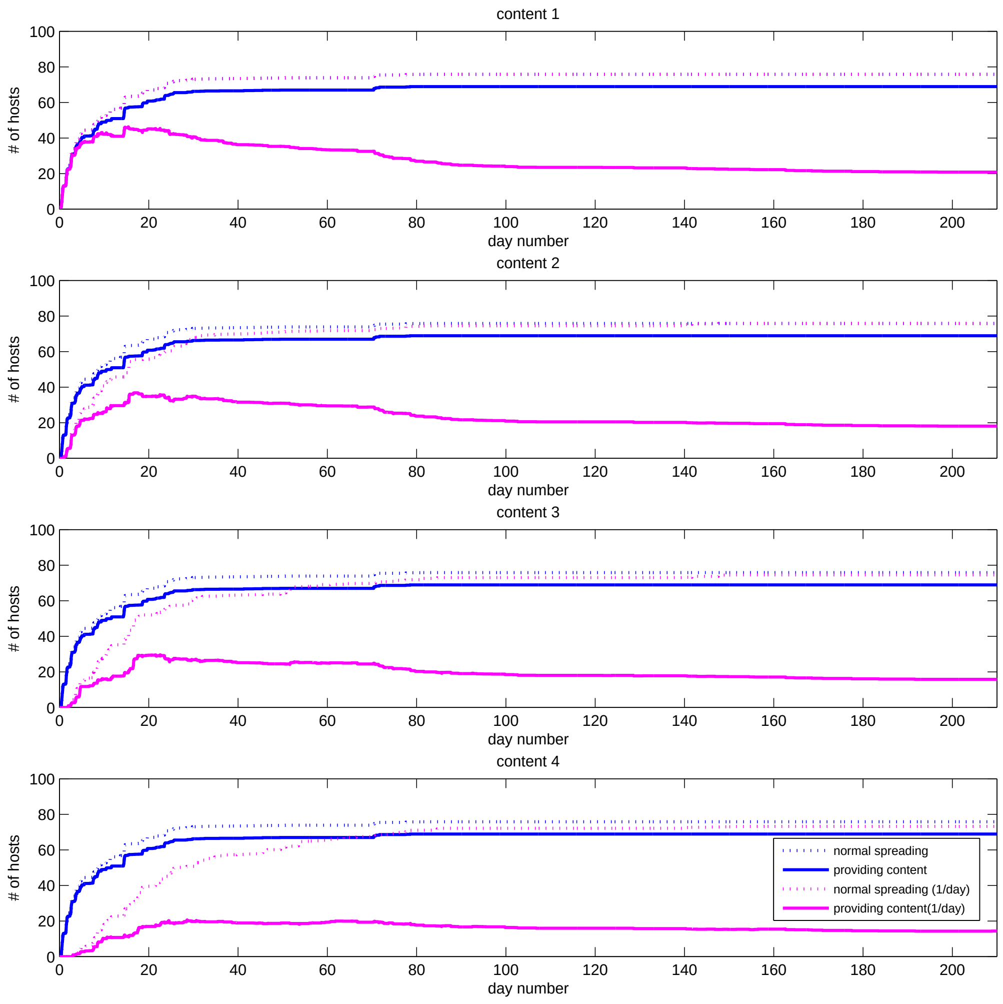


Figure C.32.: Effect of Send Rate in Average Classified MIT Weeks:*Average spreading performance of 300 contents generated by a spammer at day 0. Every user detects spam with a recognition probability of 10% and exchanges that information with others. Received information is accepted after exceeding a suggestion threshold of 10 opinions. Without send rate of 1 content/day, all 300 contents are rated only once at first reception. When using the send rate, every user rates received content every day after reception. The graph shows the people providing the first four contents of the burst.*

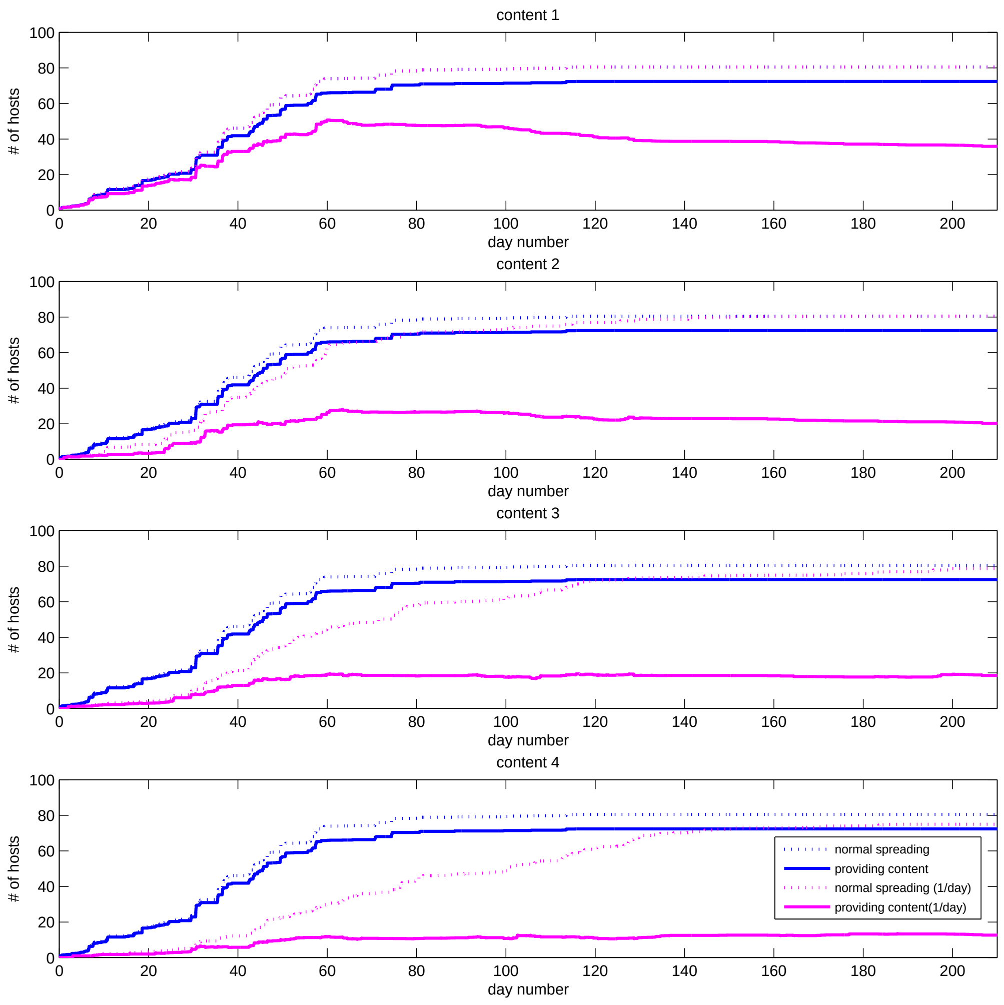


Figure C.33.: Effect of Send Rate in Bad Classified MIT Weeks:*Average spreading performance of 300 contents generated by a spammer at day 0. Every user detects spam with a recognition probability of 10% and exchanges that information with others. Received information is accepted after exceeding a suggestion threshold of 10 opinions. Without send rate of 1 content/day, all 300 contents are rated only once at first reception. When using the send rate, every user rates received content every day after reception. The graph shows the people providing the first four contents of the burst.*

C.6. Personal vs. Local Blacklist

	Personal		Threshold 20		Threshold 10	
time	having	providing	having	providing	having	providing
1 week	95.51%	85.90%	95.51%	85.90%	95.51%	85.90%
2 weeks	86.74%	75.99%	86.74%	75.99%	86.74%	75.99%
4 weeks	89.11%	79.23%	89.11%	79.23%	89.11%	79.23%
8 weeks	91.36%	72.29%	91.36%	72.29%	91.16%	65.65%
16 weeks	93.68%	62.29%	93.51%	60.54%	92.92%	54.11%

Table C.16.: Comparison Personal and Local Blacklist in Bad Classified Weeks: *Malicious Content is generated every day and exchanged among users which detect spam with a recognition probability of 10%. Additionally, local blacklists with a suggestion threshold of 20 and 10 opinions are used.*

	Personal		Threshold 20		Threshold 10	
time	having	providing	having	providing	having	providing
1 week	99.41%	84.98%	94.41%	84.98%	99.41%	84.98%
2 weeks	97.32%	77.03%	97.32%	77.03%	96.78%	72.21%
4 weeks	99.34%	67.93%	99.34%	67.39%	99.28%	57.52%
8 weeks	99.94%	56.30%	99.94%	53.12%	99.53%	46.17%
16 weeks	98.34%	42.51%	98.34%	36.26%	98.16%	30.98%

Table C.17.: Comparison Personal and Local Blacklist in Average Classified Weeks: *Malicious Content is generated every day and exchanged among users which detect spam with a recognition probability of 10%. Additionally, local blacklists with a suggestion threshold of 20 and 10 opinions are used.*

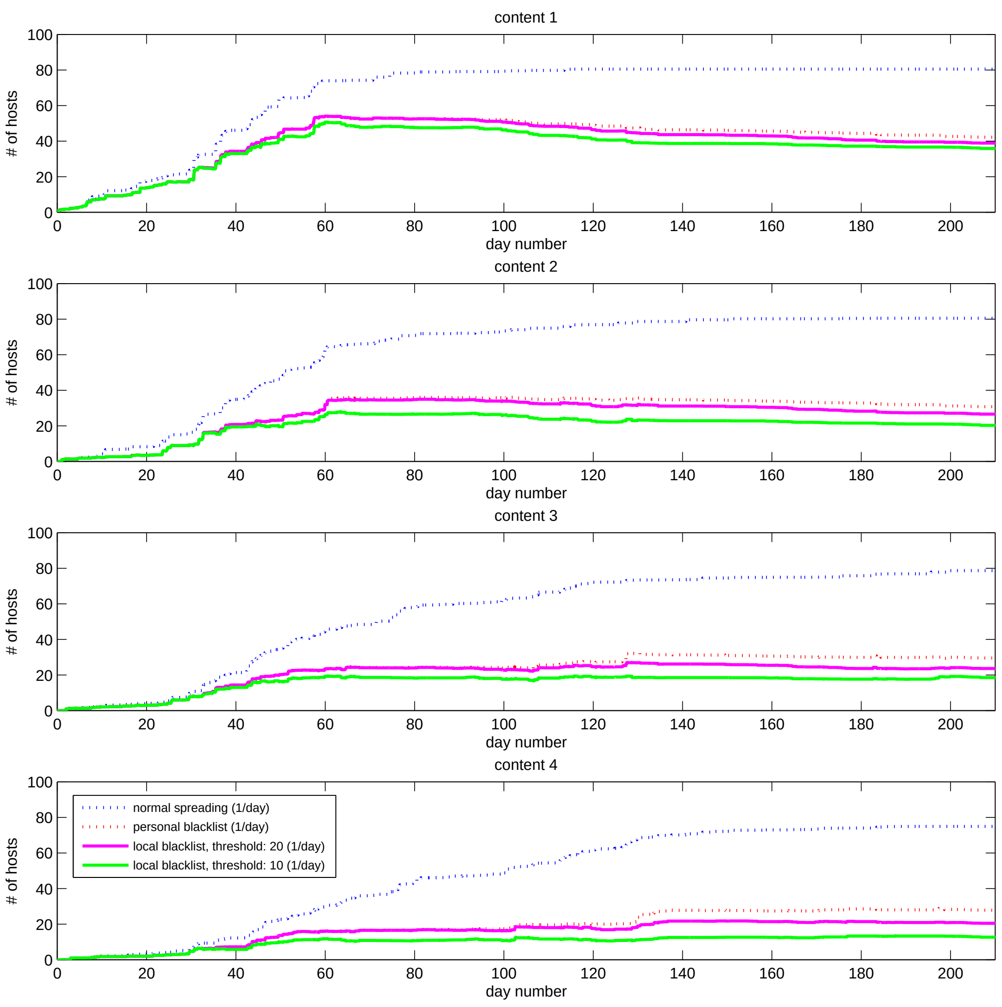


Figure C.34.: Comparison of Personal and Local Blacklists in Bad Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every day. Personal and Local Blacklisting is combined with a send rate limitation of 1 content/day. Every user individually detects spam with a recognition probability of 10%. Additionally, suggestion thresholds of 20 and 10 opinions are used. The graph shows the percentages of people providing the first four contents generated in the first four weeks.*

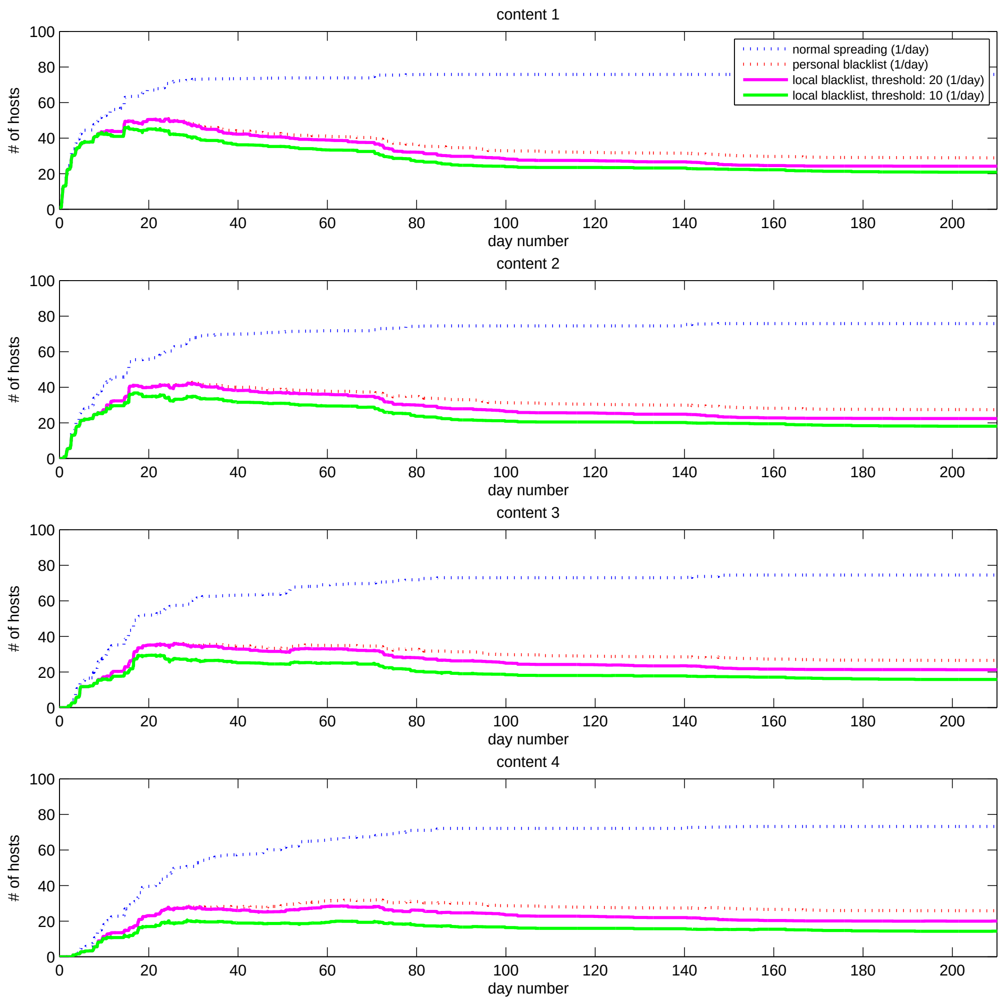


Figure C.35.: Comparison of Personal and Local Blacklists in Average Classified MIT Weeks: *Average spreading performance of one content generated by a spammer every day. Personal and Local Blacklisting is combined with a send rate limitation of 1 content/day. Every user individually detects spam with a recognition probability of 10%. Additionally, suggestion thresholds of 20 and 10 opinions are used. The graph shows the percentages of people providing the first four contents generated in the first four weeks.*

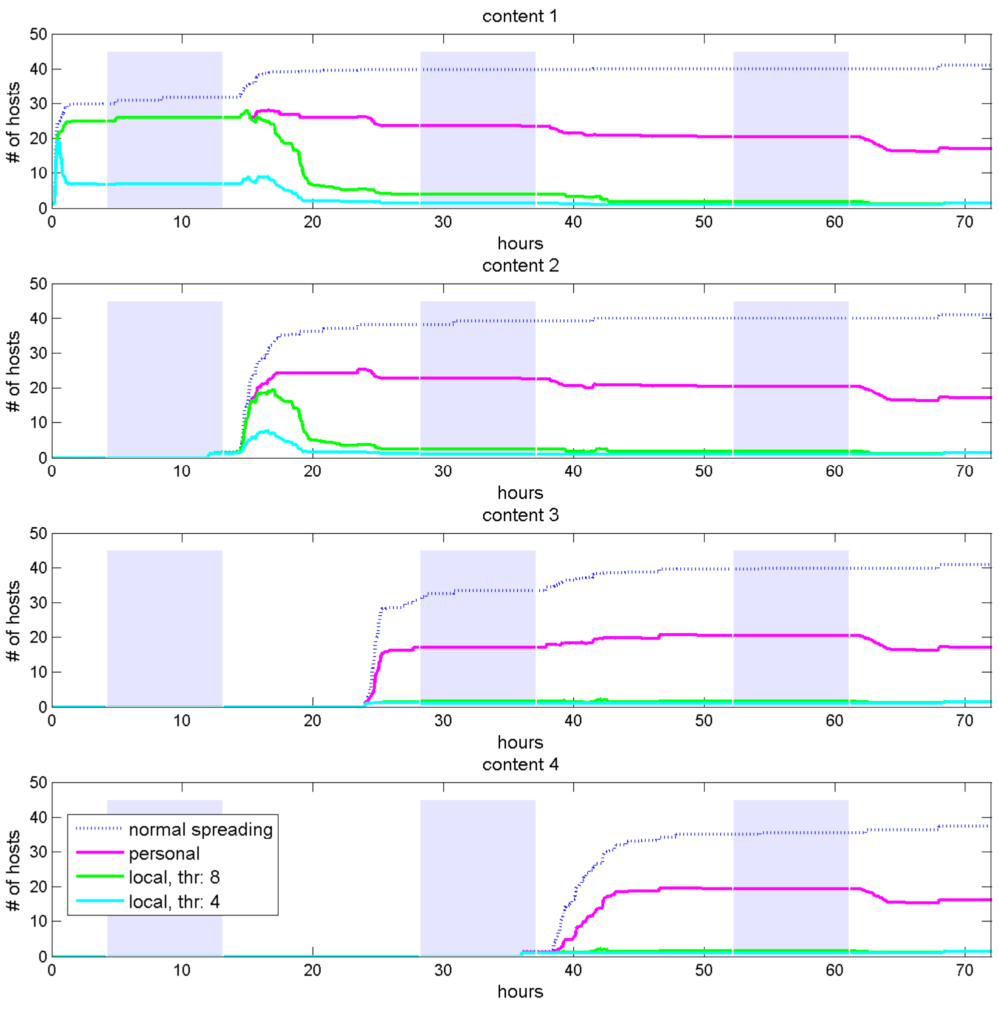


Figure C.36.: Comparison of the Personal and Local Blacklists in the Haggle Set: *Average spreading performance of a content generated by a spammer every 12 hours. The percentage of people providing content is compared between Personal and Local Blacklisting. Every user individually detects spam with a recognition probability of 20%. Additionally, suggestion thresholds of 8 (∼ 20% of total set) and 4 (∼ 10%) opinions are used. The gray shaded regions show an 8 hour inactivivity period during night. The graph shows the percentages of people providing the first four contents generated in the first 36 hours.*

C.7. Influence of Social Weights

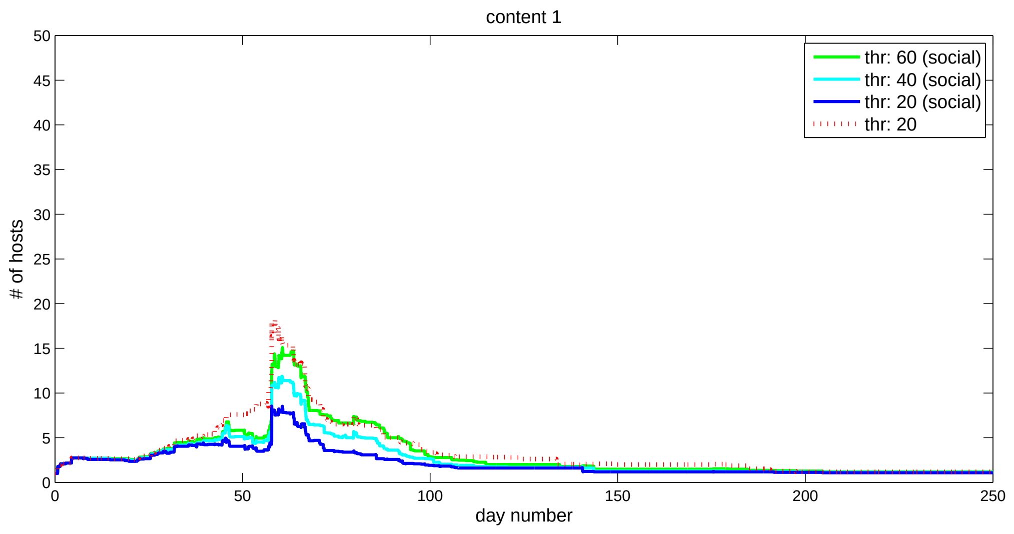


Figure C.37.: Community Weighted Recommendations: *The percentage of people providing content is compared between a fixed suggestion threshold of 20, and community weighted thresholds of 20, 30 and 40 opinions for a spam recognition of 20%.*

D. PodNet Implementation

D.1. Existing Protocol

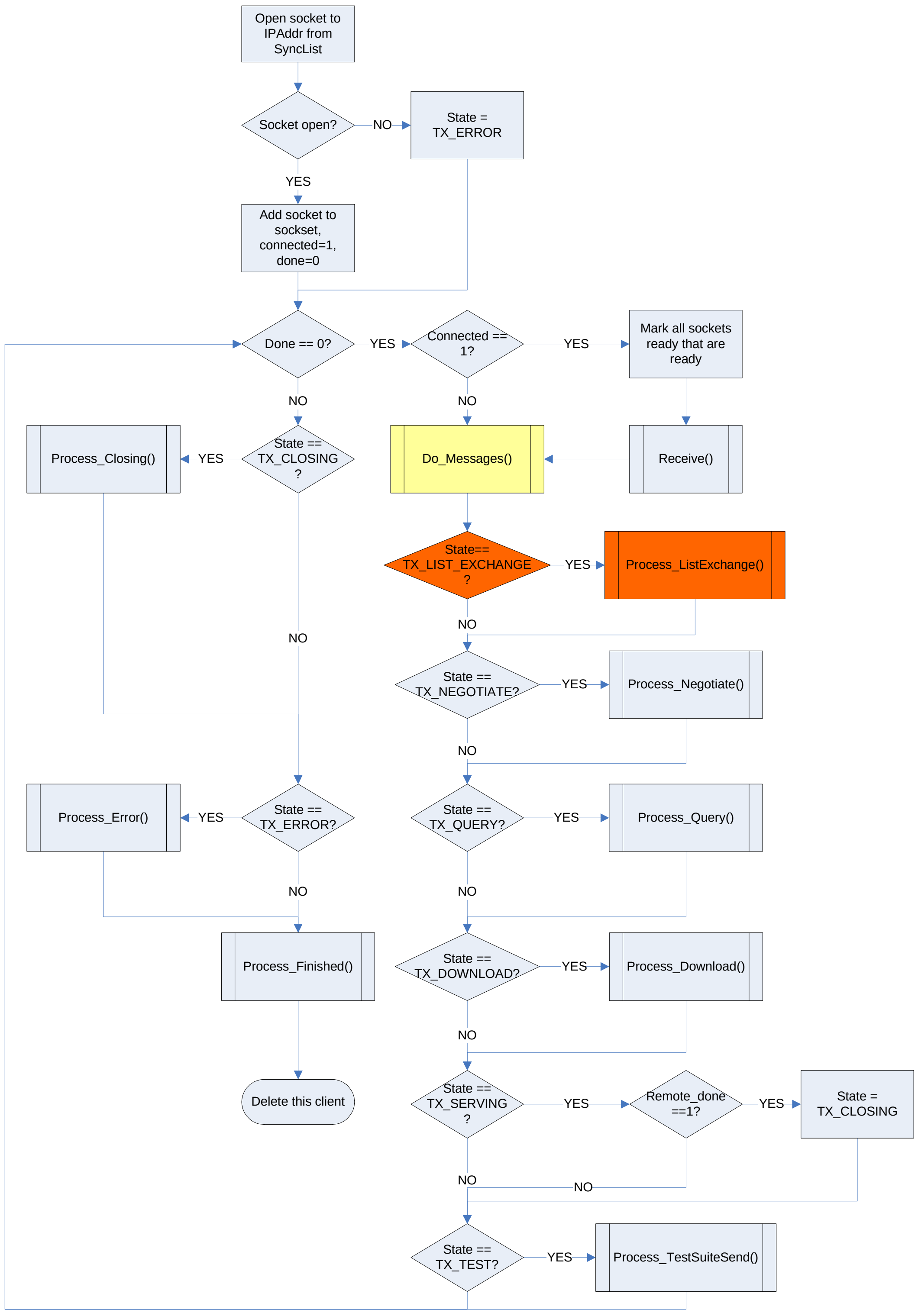


Figure D.1.: Transer_Client::ThreadRun(): *The red colored block is new.*

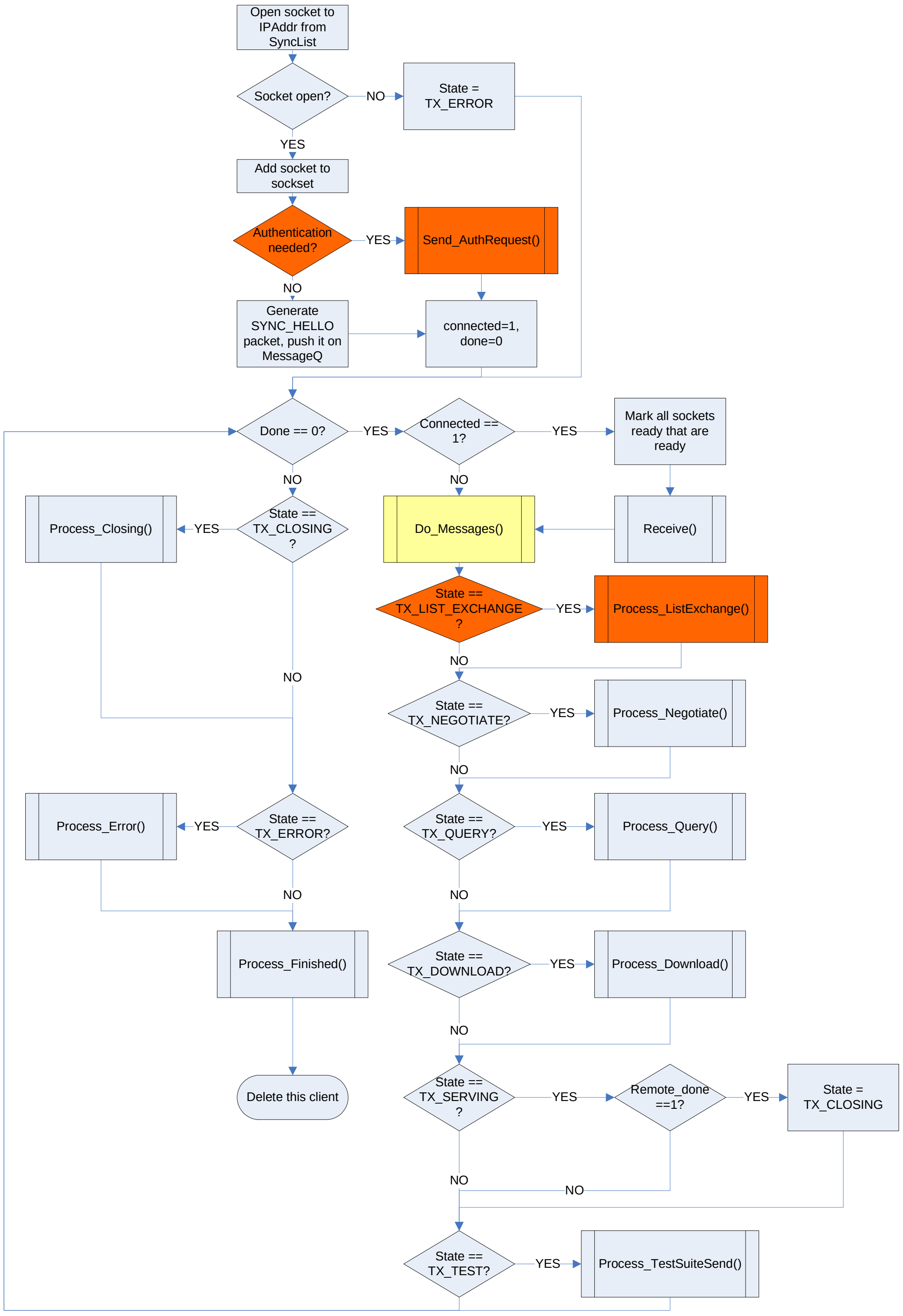


Figure D.2.: Transfer_Server::ThreadRun(): *The red colored block is new.*

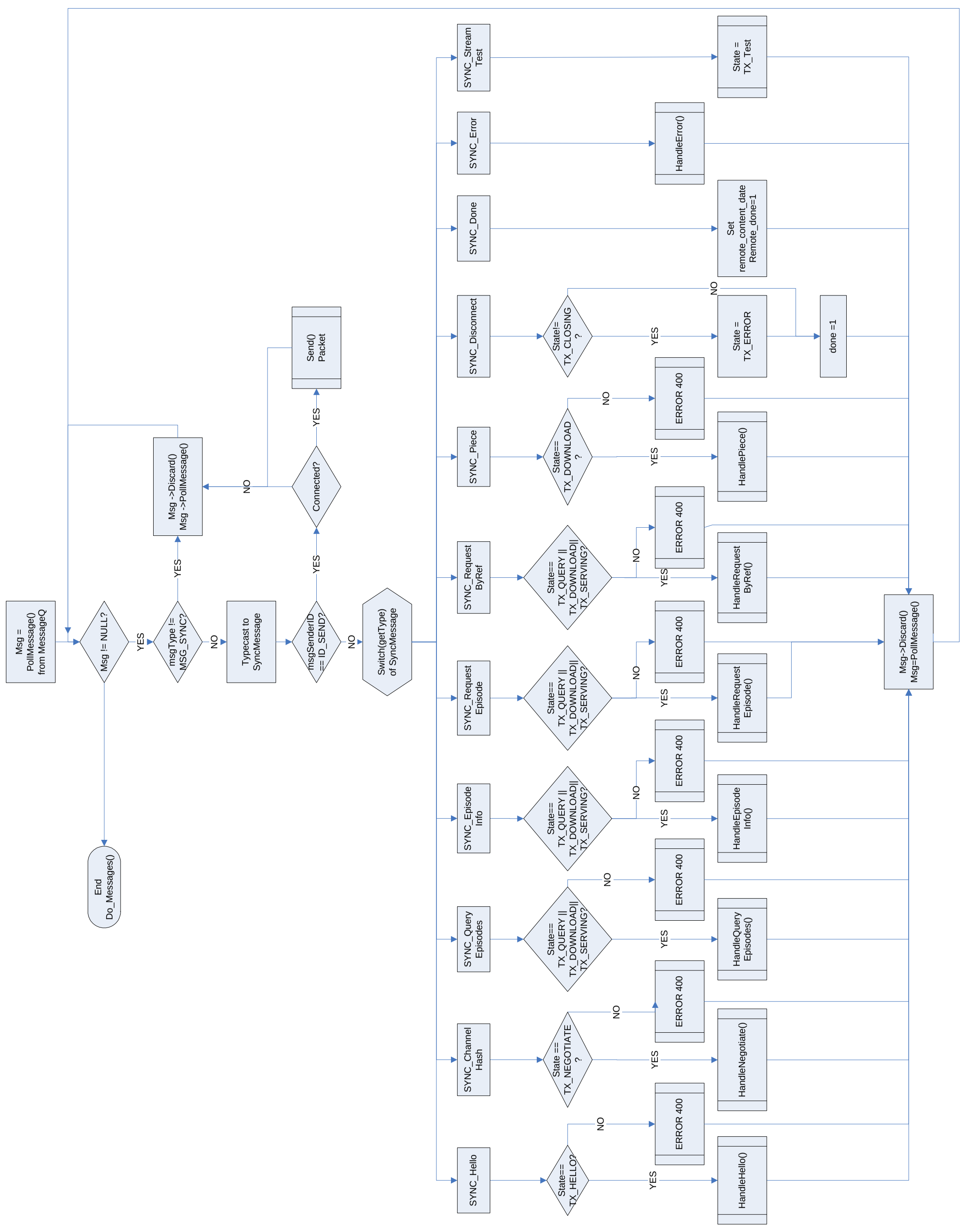


Figure D.3.: Transfer_Common::Do_Messages(): *This part is still the same than in the original implementation.*

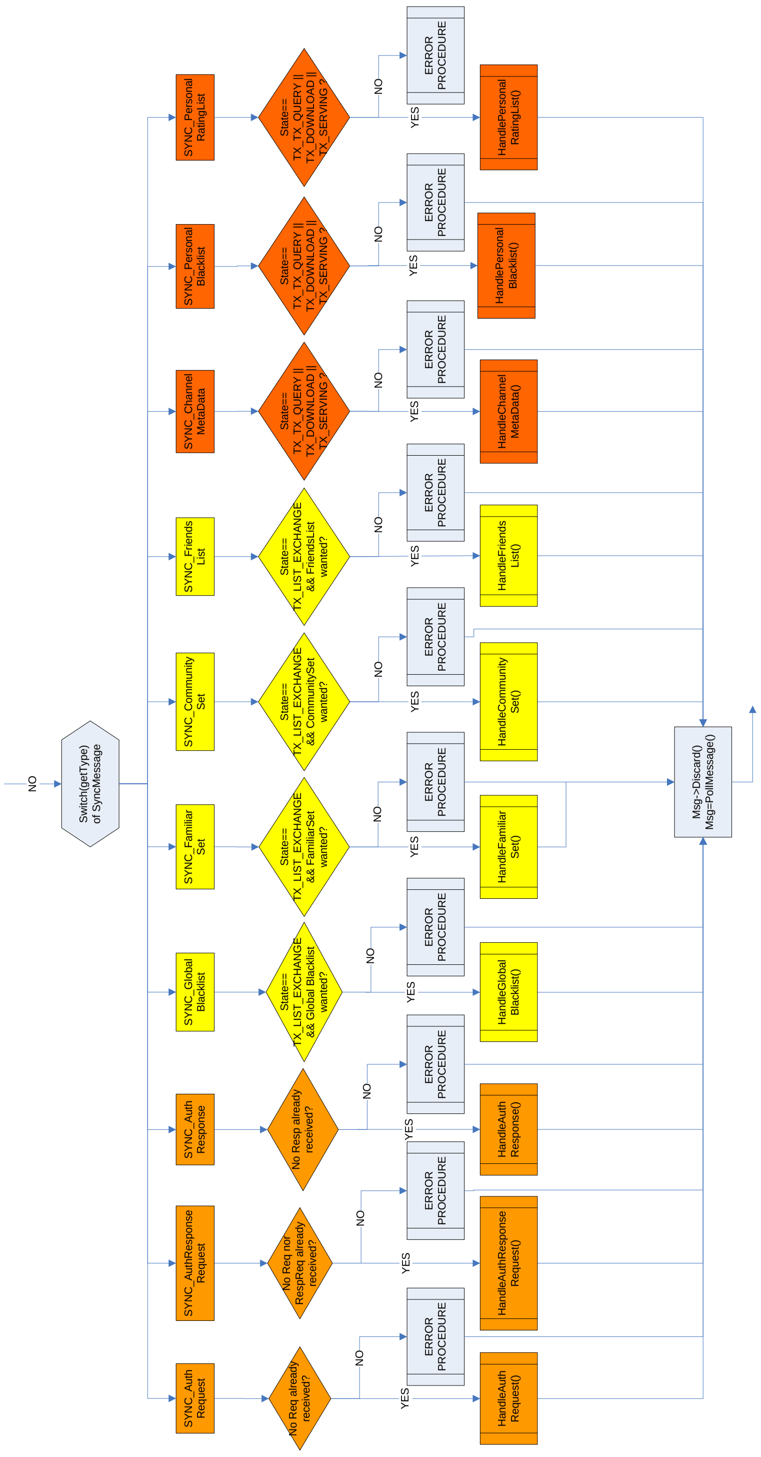

NO
Switch(getType) of SyncMessage
SYNC_Auth Request
SYNC_AuthResponse Request
SYNC_Auth Response
SYNC_Global Blacklist
SYNC_Familiar Set
SYNC_Community Set
SYNC_Friends List
SYNC_Channel MetaData
SYNC_Personal Blacklist
SYNC_Personal RatingList
No Req already received?
No Req nor RespReq already received?
No Resp already received?
State== TX_LIST_EXCHANGE && Global Blacklist wanted?
State== TX_LIST_EXCHANGE && FamiliarSet wanted?
State== TX_LIST_EXCHANGE && CommunitySet wanted?
State== TX_LIST_EXCHANGE && FriendsList wanted?
State== TX_TX_QUERY || TX_DOWNLOAD || TX_SERVING ?
State== TX_TX_QUERY || TX_DOWNLOAD || TX_SERVING ?
State== TX_TX_QUERY || TX_DOWNLOAD || TX_SERVING ?
NO
YES
ERROR PROCEDURE
HandleAuth Request()
HandleAuthResponse Request()
HandleAuth Response()
HandleGlobal Blacklist()
HandleFamiliar Set()
HandleCommunity Set()
HandleFriends List()
HandleChannel MetaData()
HandlePersonal Blacklist()
HandlePersonal RatingList()
Msg->Discard() Msg=PollMessage()

Figure D.4.: New Part of Transfer_Common::Do_Messages(): *The yellow blocks are performed before negotiating any channels, the orange colored authentication blocks are either used at the Hello stage or just before the download stage and the red colored blocks may be used before exchanging the Episode meta data for a channel.*

E. Evaluation

E.1. Cryptographic Functions

All evaluation results from the cryptographic functions test are presented in this section. The performance tests were evaluated on a HP iPAQ and a HTC Touch. The specifications can be found in Table 5.1. The results of the performance tests on symmetric cryptography as well as asymmetric cryptography are presented in Tables E.1 - E.5. The colors in the tables correspond to different file and string sizes in Bytes. See below for color definition.

🟨 70B
🟧 190B
🟥 1900B
🟪 19000000B

iPAQ Public Key Cryptography					
	Generate Key	Encrypt String	Decrypt String	Encrypt File	Decrypt File
RSA 1024	1539	🟨 3	🟨 22	🟨 8	🟨 27
RSA 2048	8303	🟧 5	🟧 113	🟧 11	🟧 119
RSA 4096	61151	🟧 11	🟧 673	🟧 16	🟧 685
ECC 160	419	🟧 41	🟧 29	🟧 52	🟧 39
ECC 224	419	🟧 612	🟧 58	🟧 40	🟧 61
ECC 384	1196	🟧 197	🟧 131	🟧 198	🟧 159

Table E.1.: Encryption and Decryption with Asymmetric Cryptography on iPAQ: *Times in* [*ms*].

HTC Public Key Cryptography					
	Generate Key	Encrypt String	Decrypt String	Encrypt File	Decrypt File
RSA 1024	5133	🟨 8	🟨 103	🟨 52	🟨 141
RSA 2048	27784	🟧 15	🟧 475	🟧 56	🟧 522
RSA 4096	214411	🟧 33	🟧 2564	🟧 84	🟧 2625
ECC 160	1267	🟨 136	🟨 94	🟨 185	🟨 137
ECC 224	1887	🟧 193	🟧 135	🟧 242	🟧 182
ECC 384	4923	🟧 876	🟧 610	🟧 918	🟧 718

Table E.2.: Encryption and Decryption with Asymmetric Cryptography on HTC: *Times in [ms].*

iPAQ Public Key Cryptography					
	Sign String	Verify String	Sign File	Verify File	Verify Bad
RSA 1024	24	2	25	4	2
RSA 2048	113	4	116	6	4
RSA 4096	668	8	668	10	9
ECC 160	22	33	24	35	1
ECC 224	30	60	31	61	60
ECC 384	100	198	102	203	199

Table E.3.: Signing and Verifying with Asymmetric Cryptography on iPAQ: *Times in [ms].*

HTC Public Key Cryptography					
	Sign String	Verify String	Sign File	Verify File	Verify Bad
RSA 1024	103	6	103	10	7
RSA 2048	479	11	481	16	11
RSA 4096	2573	26	2584	30	27
ECC 160	70	100	72	104	3
ECC 224	102	184	104	192	187
ECC 384	342	726	442	730	730

Table E.4.: Signing and Verifying with Asymmetric Cryptography on HTC *Times in [ms].*

iPAQ Symmetric Key Cryptography					
	Generate Key	Encrypt String	Decrypt String	Encrypt File	Decrypt File
AES CBC 128	3	1	0	17	15
AES CBC 256	3	1	0	17	11

Table E.5.: Encryption and Decryption with Symetric Crypto on iPAQ: *Times in [ms].*

HTC Symmetric Key Cryptography					
	Generate Key	Encrypt String	Decrypt String	Encrypt File	Decrypt File
AES CBC 128	99	3	2	75	71
AES CBC 256	1	2	0	55	56

Table E.6.: Encryption and Decryption with Symmetric Cryptography on HTC: *Times in [ms].*

iPAQ Symmetric Cryptography (large file)		
	Encrypt File	Decrypt File
AES CBC 256	6877	6344

Table E.7.: Encryption and Decryption with Symmetric Cryptography on iPAQ: *Times in [ms].*

String Lengths								
	Private Key		Public Key		Signature		Encrypted	
	raw	hex encoded	raw	hex encoded	raw	hex encoded	raw	hex encoded
RSA 1024	631	1262	160	320	128	256	128	256
RSA 2048	1215	2430	292	584	256	512	256	512
RSA 4096	2373	4746	548	1096	512	1024	512	1024
ECC 160	203	406	214	428	42	84	251	502
ECC 224	258	516	279	558	56	112	267	534
ECC 384	401	802	441	882	96	192	307	614

Table E.8.: String Lengths in Bytes

E.2. Communication Overhead

In this section, we analyze the data transfer of two devices running on the current and the original implementation after capturing the exchanged information with Wireshark[1]. The connecting device acts as client connecting to another device which acts as server. A device only connects to another device if it assumes that the other device stores new informations. In general, the server answers TCP connection request by sending a first Hello message to the client.

E.2.1. Unknown Devices - No Channels

In this scenario two devices that do not know each other connect. Since both devices do not know each other, a three way challenge request response mechanism replaces the existing exchange of Hello messages[2] in the Hello Procedure. The messages are larger because they comprise the user credentials.
Both devices do not hold any channels or episodes and therefore only information from the empty discovery channel is exchanged. In the visualization in Table E.9, device A acts as client and wants to connect to a device B (server) in order to learn information about it. If any of these two devices would already store a global blacklist, social lists (familiar set, community set or friends list) which comprise other user's identities, this information would be exchanged between the Hello Procedure and the ChannelHash exchange as shown in Figure 5.2. In every later connection of these two devices, the lists would only be exchanged if a device would have an updated version of its own (or global) list.
In the Hello Procedure, it is evaluated whether a list update is needed since every device adds a timestamp of the last received version of the other device's social lists to the message. The other device will then decide whether it stores a more updated version which has to be exchanged. This minimizes the communication overhead since lists are only exchanged if they are needed. Note that the size of these list depend on the number of entries and we do not evaluate it here for a better comparability with the original implementation.

[1]http://www.wireshark.org/
[2]Note that if one of these two devices would know the other device, the Hello Procedure would be replaced by a two way Response Request mechanism.

Device A		**Device B**
	⟵	AuthRequest (33 Bytes)
AuthResponseRequest (1011 Bytes)	⟶	
	⟵	AuthResponse (982 Bytes)
(possible list exchange)	⟶	
	⟵	(possible list exchange)
...		...
ChannelHash (156 Bytes)	⟶	
	⟵	ChannelHash (156 Bytes)
QueryEpisodes (60 Bytes)	⟶	
	⟵	QueryEpisodes (60 Bytes)
EpisodeInfo (34 Bytes)	⟶	
	⟵	EpisodeInfo (34 Bytes)
Done (25 Bytes)	⟵	
	⟶	Done (25 Bytes)

Table E.9.: Current Implementation: Connection of two Unknow Devices

Device A		**Device B**
	⟵	Hello (26 Bytes)
Hello (26 Bytes)	⟶	
ChannelHash (156 Bytes)	⟶	
	⟵	ChannelHash (156 Bytes)
QueryEpisodes (50 Bytes)	⟶	
	⟵	QueryEpisodes (50 Bytes)
EpisodeInfo (11 Bytes)	⟶	
	⟵	EpisodeInfo (11 Bytes)
Done (25 Bytes)	⟵	
	⟶	Done (25 Bytes)

Table E.10.: Original Implementation: Connection of two Unknow Devices

E.2.2. Generation of One Channel

In this scenario, device A generated a channel, adds this information in its Discovery Channel and advertises it as new content in the discovery messages. Device B realizes that device A provides new content and connects to device A. Since the only common channel is still the discovery channel, only channel information from this channel, including general information about the new generated channel, is exchanged. Besides this channel, no data is present in the network.

Device A		**Device B**
Hello (28 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Hello (28 Bytes)
	$\longleftarrow$	ChannelHash (156 Bytes)
ChannelHash (157 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (60 Bytes)
QueryEpisodes (60 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (34 Bytes)
EpisodeInfo (603 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Done (25 Bytes)
Done (25 Bytes)	$\longrightarrow$	

Table E.11.: Current Implementation: Connection of Two Known Devices, Exchange of Discovery Channel Information for One Additional Channel

Device A		**Device B**
Hello (26 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Hello (26 Bytes)
	$\longleftarrow$	ChannelHash (156 Bytes)
ChannelHash (157 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (50 Bytes)
QueryEpisodes (50 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (11 Bytes)
EpisodeInfo (432 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Done (25 Bytes)
Done (25 Bytes)	$\longrightarrow$	

Table E.12.: Original Implementation: Connection of Two Known Devices, Exchange of Discovery Channel Information for One Additional Channel

E.2.3. Subscribing to a Channel

In this scenario, device B subscribes to the channel that device A has created in Subsection E.2.2, adds it to its local channels and advertises the new information in the discovery message. Device A will receive the discovery message and connects to device B because it stores newer information since the last connection. Device A does not realize that it already holds that information. In this connection round, both devices ask for two channels, i.e. two *QueryEpisodes* are needed. The smaller packet corresponds to the discovery channel because it comprises only basic information, the larger packet to the new user generated channel.

In the current implementation, every device stores a timestamp for every channel meta data which is exchanged in the QueryEpisodes packet. Every receiver of the packet will check whether it stores a newer version of the meta data and only sends that information if it is newer. Because device B has not received any meta data at all, device A sends that information including the channel credentials.

Additionally with the channel timestamp, a personal timestamp which defines the last version of personal list received from the other device would be exchanged. A device that receives the timestamp will check whether it has updated its own personal lists in the meantime and only sends the new list if it is newer. Note that in the example below, no personal lists (personal blacklist and personal rating lists) are exchanged because they are still all empty. We did not simulate the personal list exchange for a better comparability with the original implementation and since the size of these lists depend on the number of entries.

Device A		**Device B**
	$\longleftarrow$	Hello (28 Bytes)
Hello (28 Bytes)	$\longrightarrow$	
	$\longleftarrow$	ChannelHash (157 Bytes)
ChannelHash (157 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (60 Bytes)
QueryEpisodes (60 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (63 Bytes)
QueryEpisodes (63 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (603 Bytes)
EpisodeInfo (603 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (16 Bytes)
ChannelMetaData Part1 (1460 Bytes)	$\longrightarrow$	
ChannelMetaData Part2 (113 Bytes)	$\longrightarrow$	
(possible personal lists)	$\longrightarrow$	
...		
EpisodeInfo (34 Bytes)	$\longrightarrow$	
Done (25 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Done (25 Bytes)

Table E.13.: Current Implementation: Connection of Two Known Devices, Exchange of Additional Channel Meta Data for One Channel

Device A		Device B
	$\longleftarrow$	Hello (26 Bytes)
Hello (26 Bytes)	$\longrightarrow$	
	$\longleftarrow$	ChannelHash (157 Bytes)
ChannelHash (157 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (50 Bytes)
QueryEpisodes (50 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (53 Bytes)
QueryEpisodes (53 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (432 Bytes)
EpisodeInfo (461 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (11 Bytes)
EpisodeInfo (11 Bytes)	$\longrightarrow$	
Done (25 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Done (25 Bytes)

Table E.14.: Original Implementation: Connection of two known Devices, exchange of Channel Information for One Channel

E.2.4. Exchanging Content

In this scenario, device A additionally generates an episode consisting of a picture file[3]. Compared to Subsection E.2.3, the channel meta data will not be exchanged anymore, since device B already stores the newest version. Device B starts a connection to device A after realizing that device A carries new information. In the Table E.16 it becomes evident that the original implementation wrongly assigned the maximum packet size to 1468 instead of 1460 bytes. As a result, TCP splits each packet into two packets, a large packet of 1460 and a small one of only 8 Bytes which slows down the transmit rate in the original implementation. In the current implementation, we observe an additional one way challenge response mechanism before data transmission as shown in Table E.15. A downloader always checks the authenticity of a supplier before starting any download. If a user only supplies content, an authentication from the downloader is not needed because the supplier will not download any potential malicious or intentionally corrupted content.
In contrast to the authentication messages in Subsection E.2.1, no credentials are exchanged because both devices already know each other from a previous session and thus only need to authenticate if they supply content. Note that if a device has already performed the authentication in the Hello Procedure of the same connection, no authentication is needed anymore before the data download.

Device A		Device B
Hello (28 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Hello (28 Bytes)
	$\longleftarrow$	ChannelHash (157 Bytes)
ChannelHash (157 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (60 Bytes)
QueryEpisodes (60 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (63 Bytes)
QueryEpisodes (63 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (603 Bytes)
EpisodeInfo (603 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (34 Bytes)
EpisodeInfo Part1 (1460 Bytes)	$\longrightarrow$	
EpisodeInfo Part2 (463 Bytes)	$\longrightarrow$	
Done (25 Bytes)	$\longrightarrow$	
	$\longleftarrow$	AuthRequest (8 Bytes)
AuthResponse (134 Bytes)	$\longrightarrow$	
	$\longleftarrow$	RequestEpisode (54 Bytes)
	$\longleftarrow$	RequestByRef (23 Bytes)
		... (19 $\times$)
Piece (1460 Bytes)	$\longrightarrow$	
... (18 $\times$)	$\longrightarrow$	
Piece (695 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Done (25 Bytes)

Table E.15.: Current Implementation: Connection of Two Known Devices, Exchange of One Content

[3]We exchanged a picture file of size of 25.95KB

Device A		Device B
Hello (26 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Hello (26 Bytes)
	$\longleftarrow$	ChannelHash (157 Bytes)
ChannelHash (157 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (50 Bytes)
QueryEpisodes (50 Bytes)	$\longrightarrow$	
	$\longleftarrow$	QueryEpisodes (33 Bytes)
QueryEpisodes (53 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (432 Bytes)
EpisodeInfo (461 Bytes)	$\longrightarrow$	
	$\longleftarrow$	EpisodeInfo (11 Bytes)
EpisodeInfo (517 Bytes)	$\longrightarrow$	
Done (25 Bytes)	$\longrightarrow$	
	$\longleftarrow$	RequestEpisode (44 Bytes)
	$\longleftarrow$	RequestByRef (23 Bytes)
		. . . (19 ×)
Piece (1460 Bytes)	$\longrightarrow$	
Piece (8 Bytes)	$\longrightarrow$	
. . . (both 18 ×)		
Piece (551 Bytes)	$\longrightarrow$	
	$\longleftarrow$	Done (25 Bytes)

Table E.16.: Original Implementation: Connection of Two Known Devices, Exchange of One Content

E.2.5. Secure Pairing

In this scenario, device A starts a secure pairing with device B. The message size are similar than in Subsection E.2.1 because they also comprise the credentials, but additionally to the authentication, a friend signature is exchanged. After completion of this mechanism, both devices add each other's friend signature on it's friends list.

Device A		**Device B**
	$\longleftarrow$	AuthRequest (928 Bytes)
AuthResponseRequest (1058 Bytes)	$\longrightarrow$	
	$\longleftarrow$	AuthResponse (134 Bytes)
	$\longleftarrow$	PairingDone (3 Bytes)
PairingDone (3 Bytes)	$\longrightarrow$	

Table E.17.: Current Implementation: Secure Pairing between Devices A and B

E.3. Computational Overhead

Processing Time Overhead			
	Original	Open Channel	Closed Channel
No File	602ms	612ms	612ms
Small File	1176ms	1345ms · hash: 84ms	1759ms · hash: 110ms · decrypt: 270ms
Large File	59066ms	63947ms · hash: 11951ms	88813ms · hash: 11778ms · decrypt: 27878ms

Table E.18.: Processing Time Overhead for No file, Small File (52.79KB) and Big File (5.99MB): *The times are averaged over 100 connections and file exchanges.*

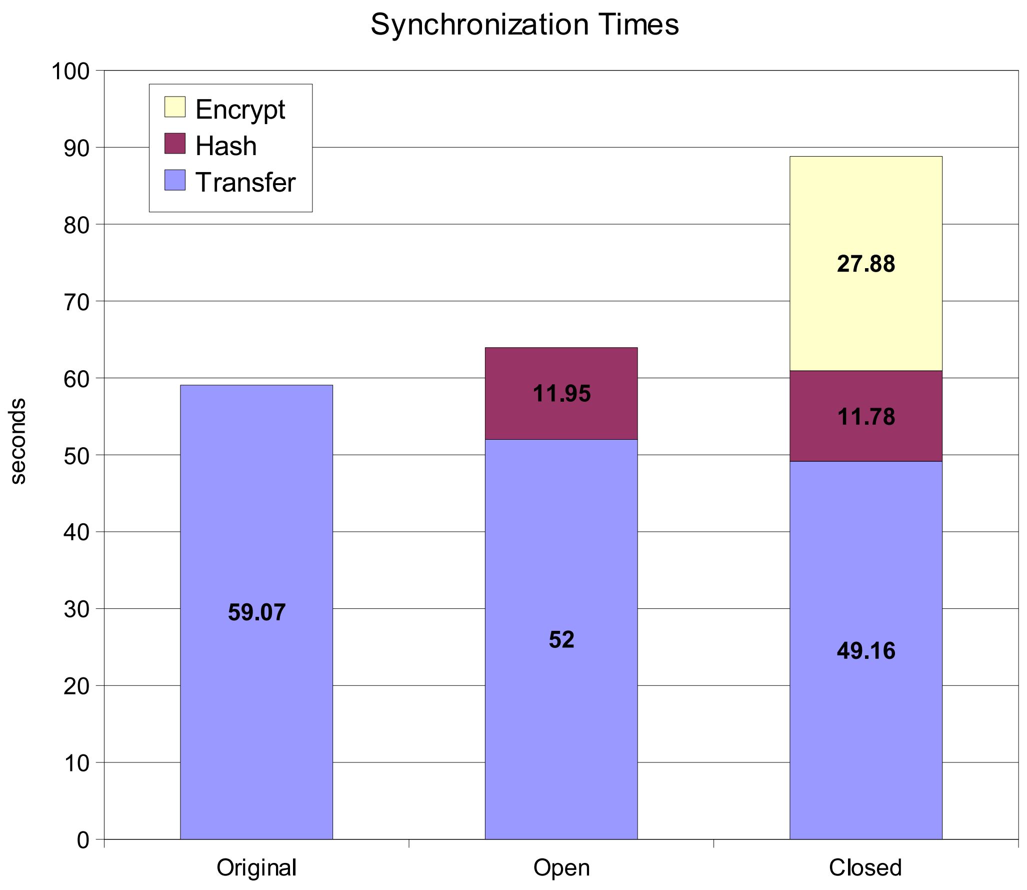


Figure E.1.: Computational Overhead Sending a Large File

E.4. Community Detection

E.4.1. MIT Set

Sloan Business School	**MIT Media Lab**		
	Freshmen	**Professor**	**Other**
Blue Tones	Green	Yellow	Red Tones

Table E.19.: Color Information for the MIT Data Set Classification.

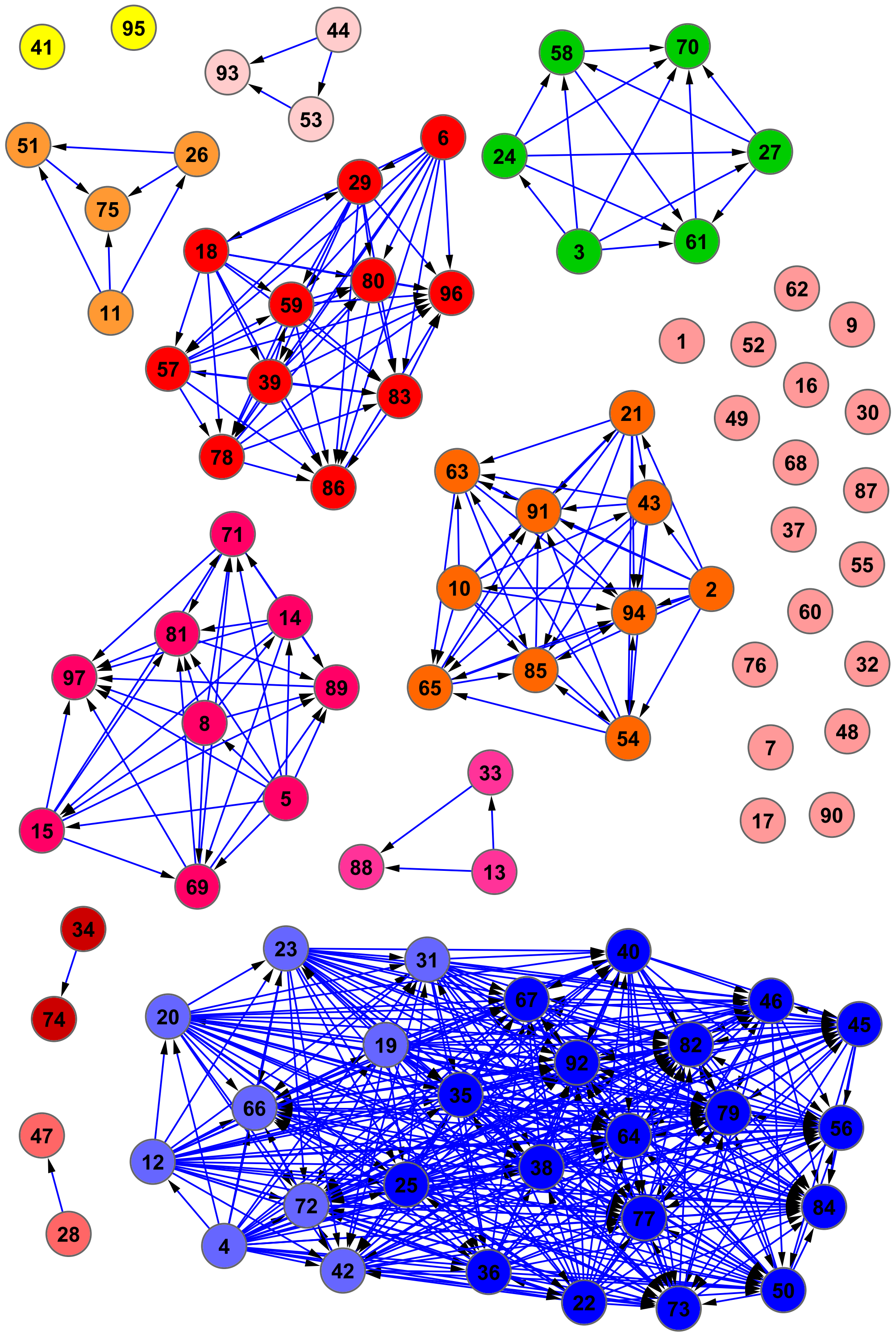


Figure E.2.: Classification of the People in the MIT Data Set: *Color Information in Table E.19.*

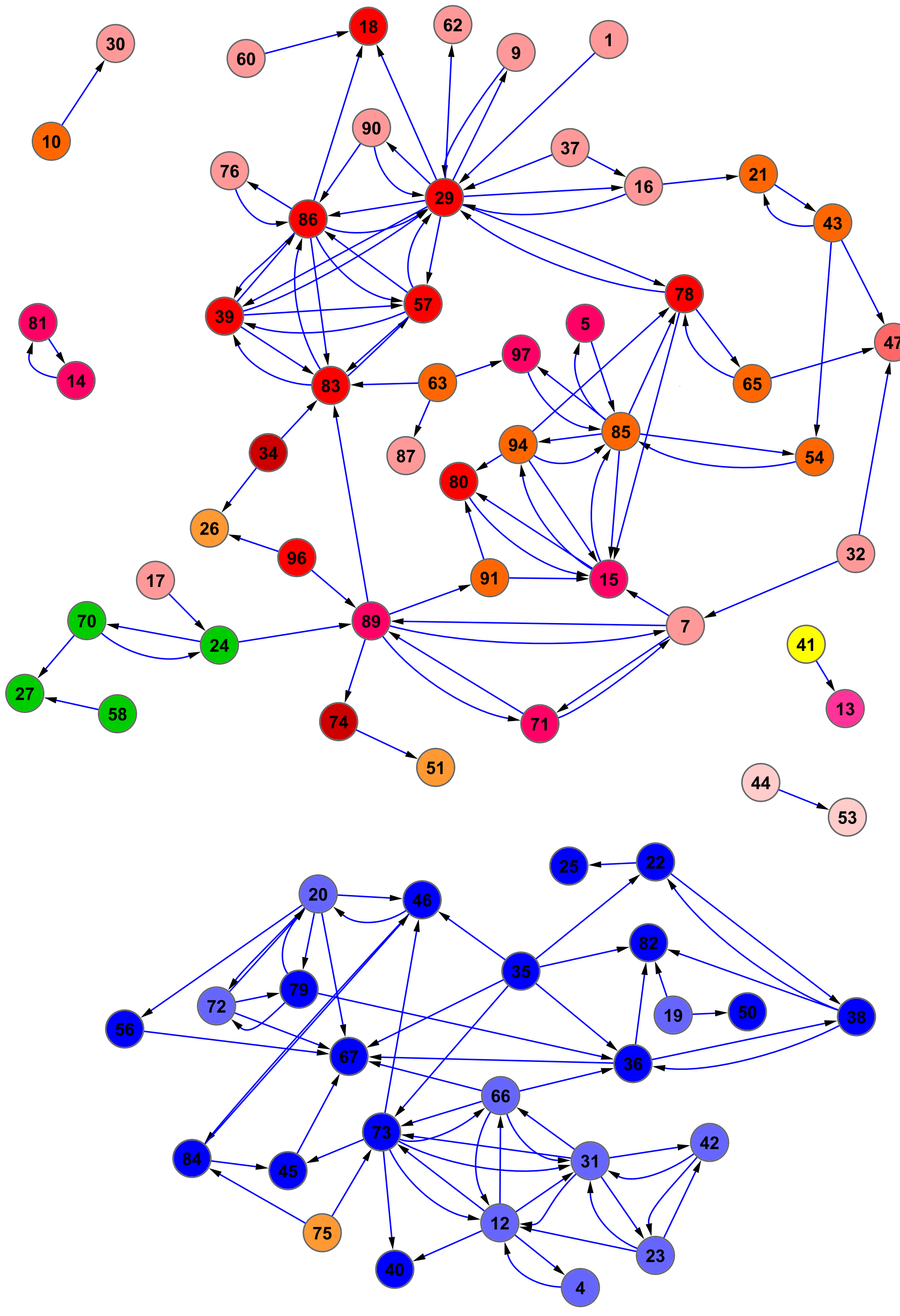


Figure E.3.: Call Connections Between People in the MIT Data Set: *Color Information in Table E.19.*

E.4.2. Aging Parameters

Fixed Parameter	**Value**
Min. familiar set size	5
Max. familiar set size	30
Min. community set size	5
Max. community set size	30
Add ratio λ	0.6
Merge ratio γ	0.6
Min. familiar set aging speed	$1\frac{s}{h}$
Max. familiar set aging speed	$40\frac{s}{h}$
Familiar set aging speed step	$0.5\frac{s}{h}$
Min. familiar set adding threshold	$600s$
Max. familiar set adding threshold	$3600s$
Familiar set adding threshold step	$200s$
Community set aging speed	real time
Max. community set removing threshold	2419200s (28days)
Community set removing threshold step	43200s (12h)

Table E.20.: Fixed Community Detection Parameters used on MIT Set.

Dynamic Parameter	**Initial Value**
Familiar aging speed	Min. familiar set aging speed
Familiar set adding threshold	Min. familiar set adding threshold
Familiar set removing threshold	Familiar set adding threshold - $200s$
Community set removing threshold	Max. community set removing threshold

Table E.21.: Community Detection Dynamic Parameter Initialization used on MIT Set.

E.4.3. Snapshots

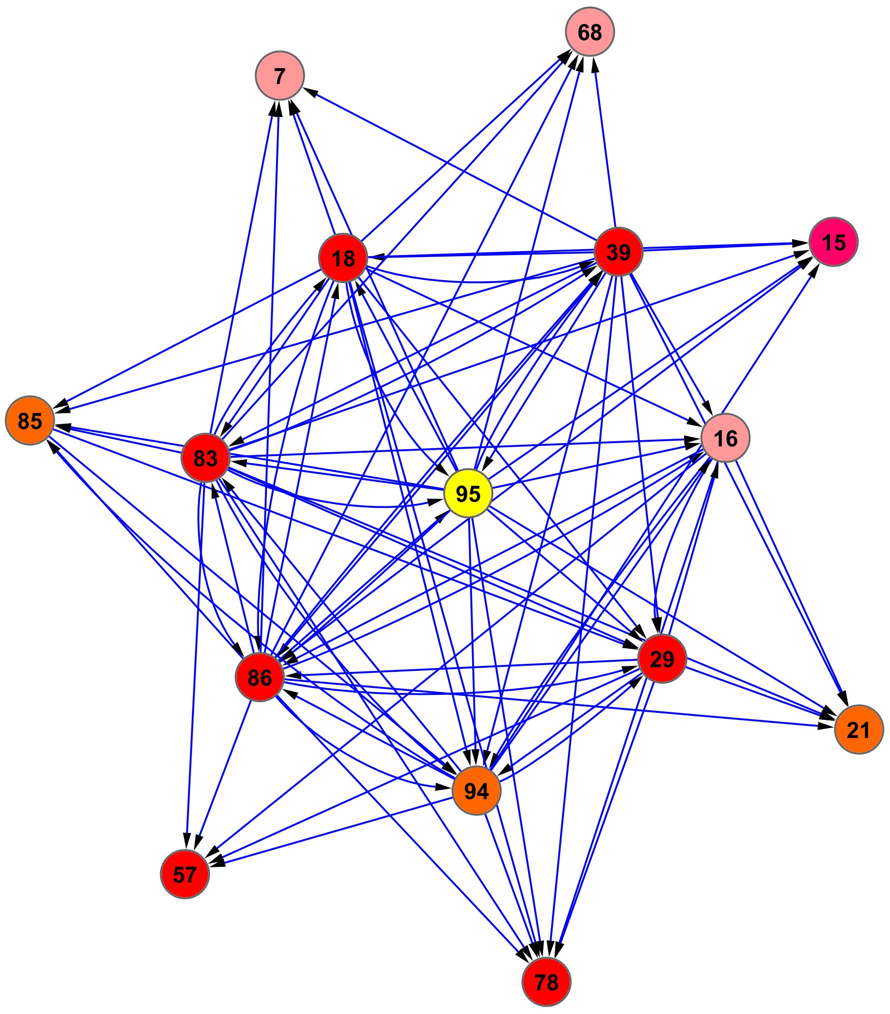


Figure E.4.: Community Graph After 4 Weeks: *Color Information in Table E.19.*

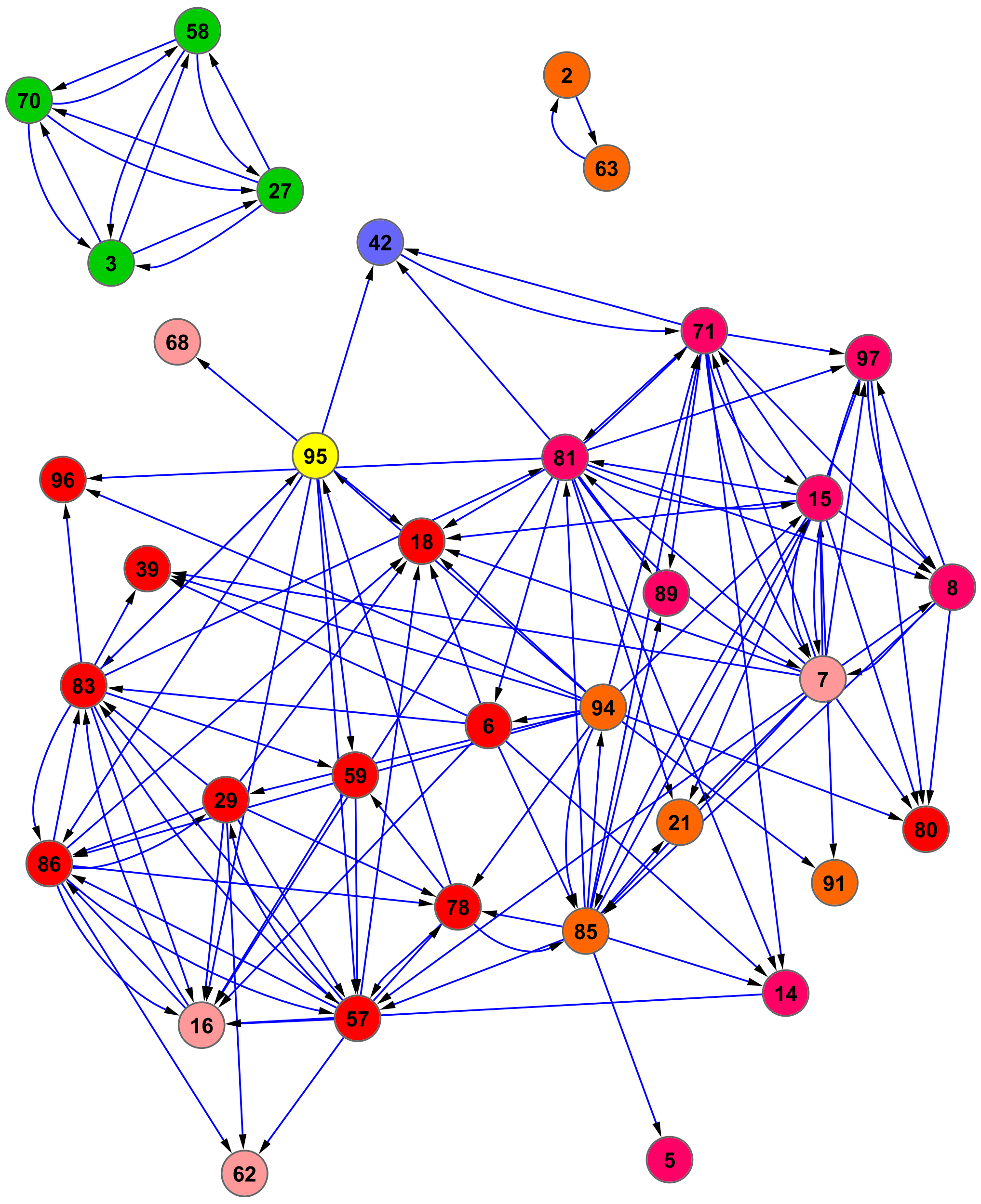


Figure E.5.: Community Graph After 8 Weeks: *Color Information in Table E.19.*

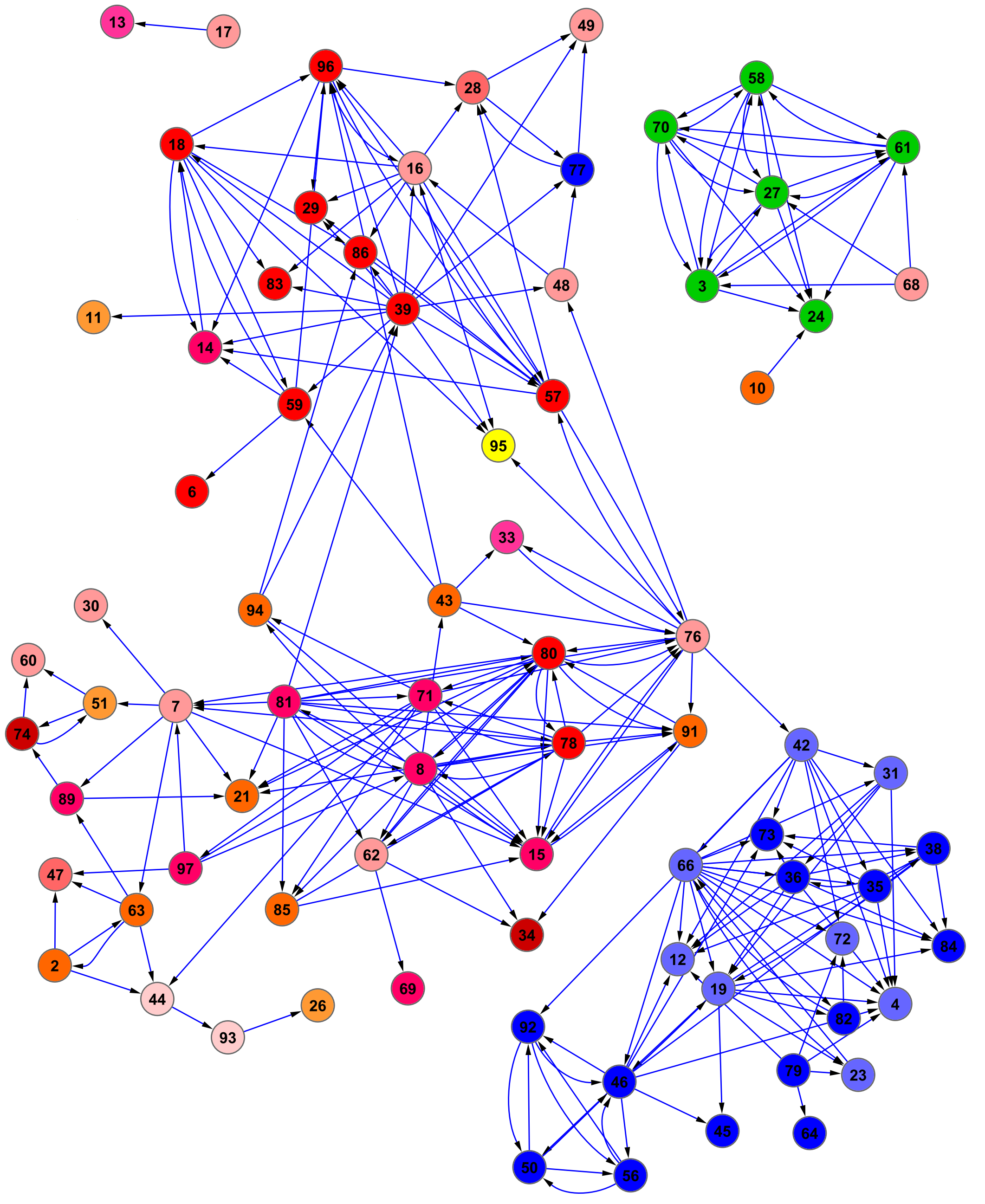


Figure E.6.: Community Graph After 12 Weeks: *Color Information in Table E.19.*

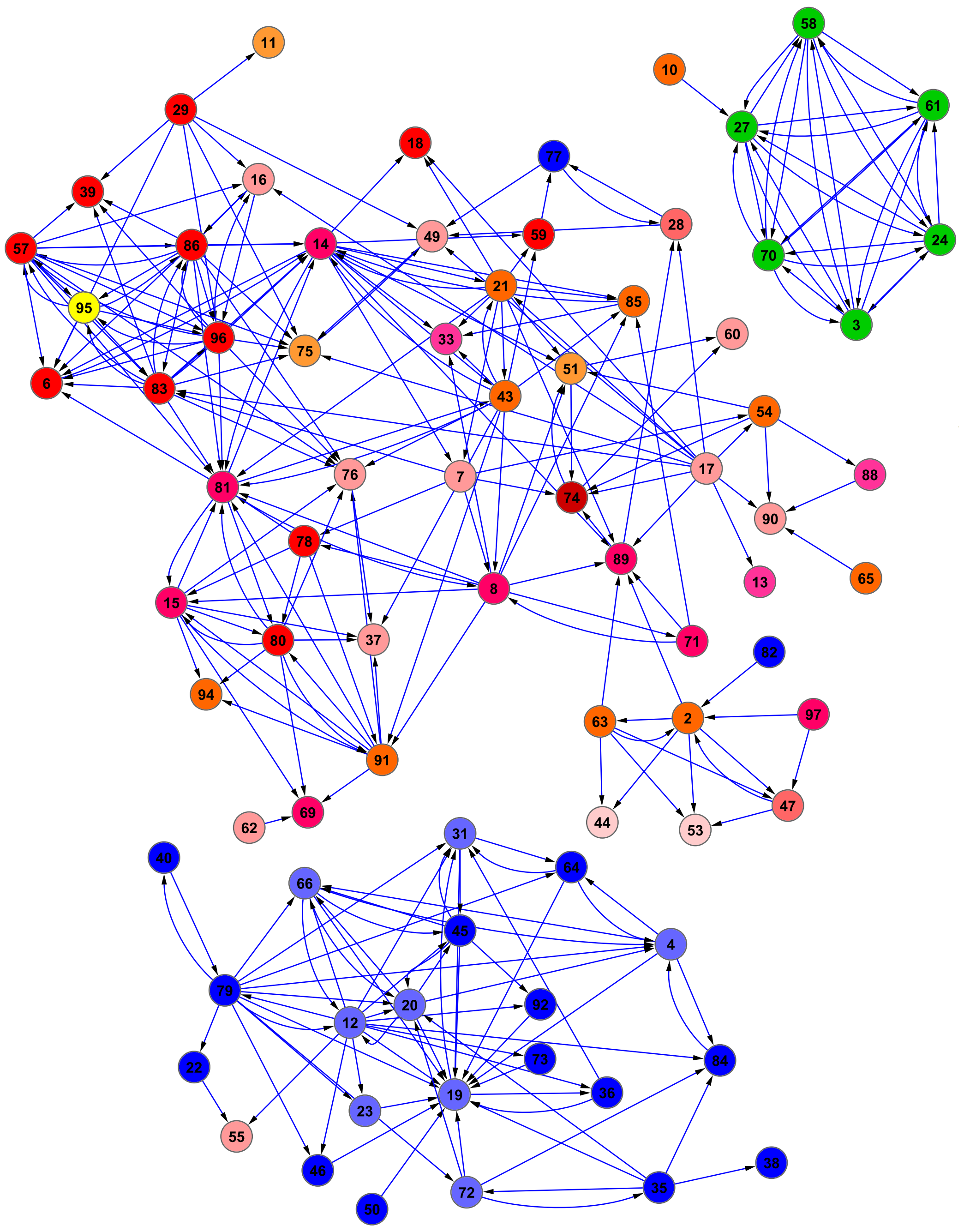


Figure E.7.: Community Graph After 16 Weeks: *Color Information in Table E.19.*

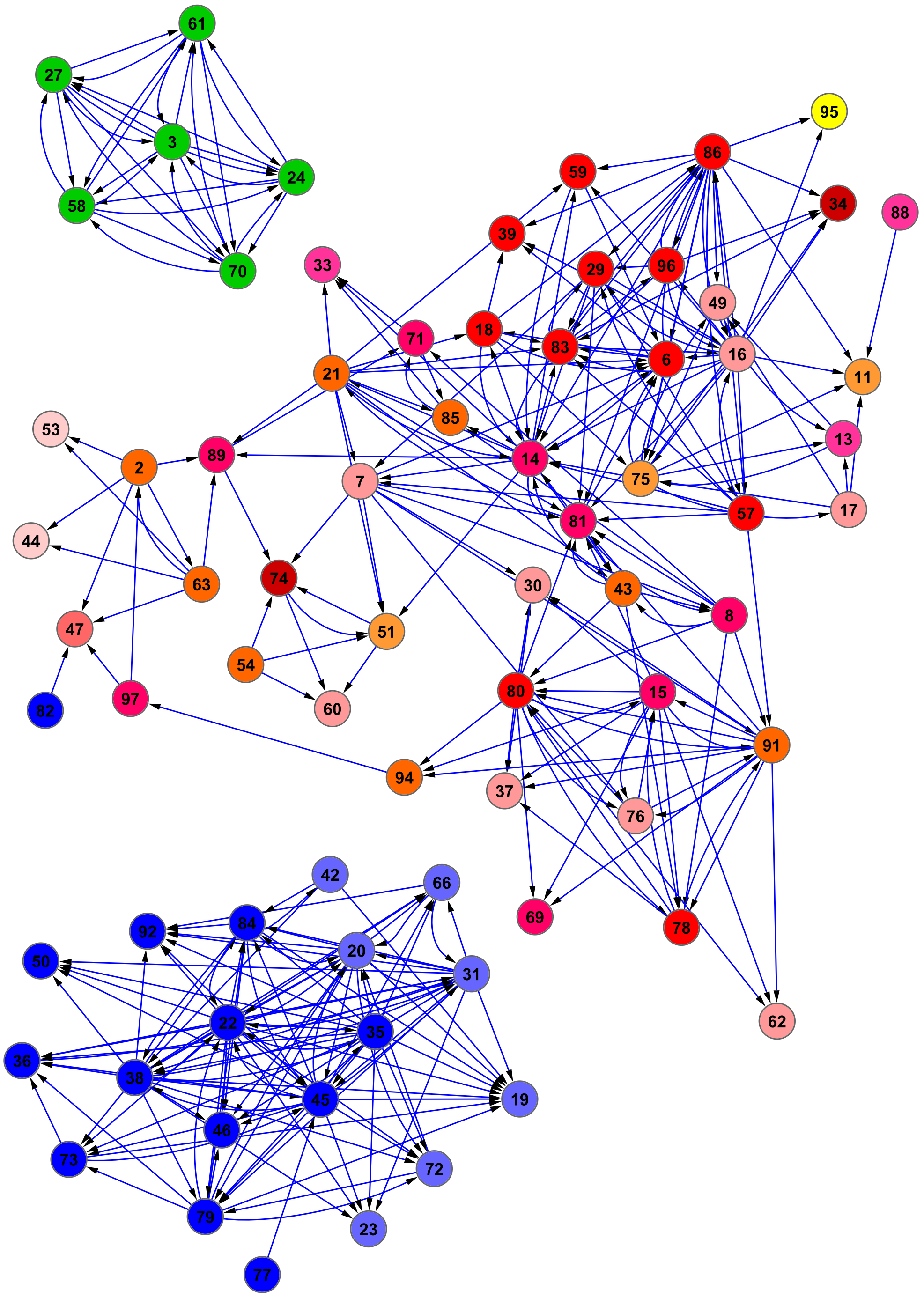


Figure E.8.: Community Graph After 20 Weeks: *Color Information in Table E.19.*

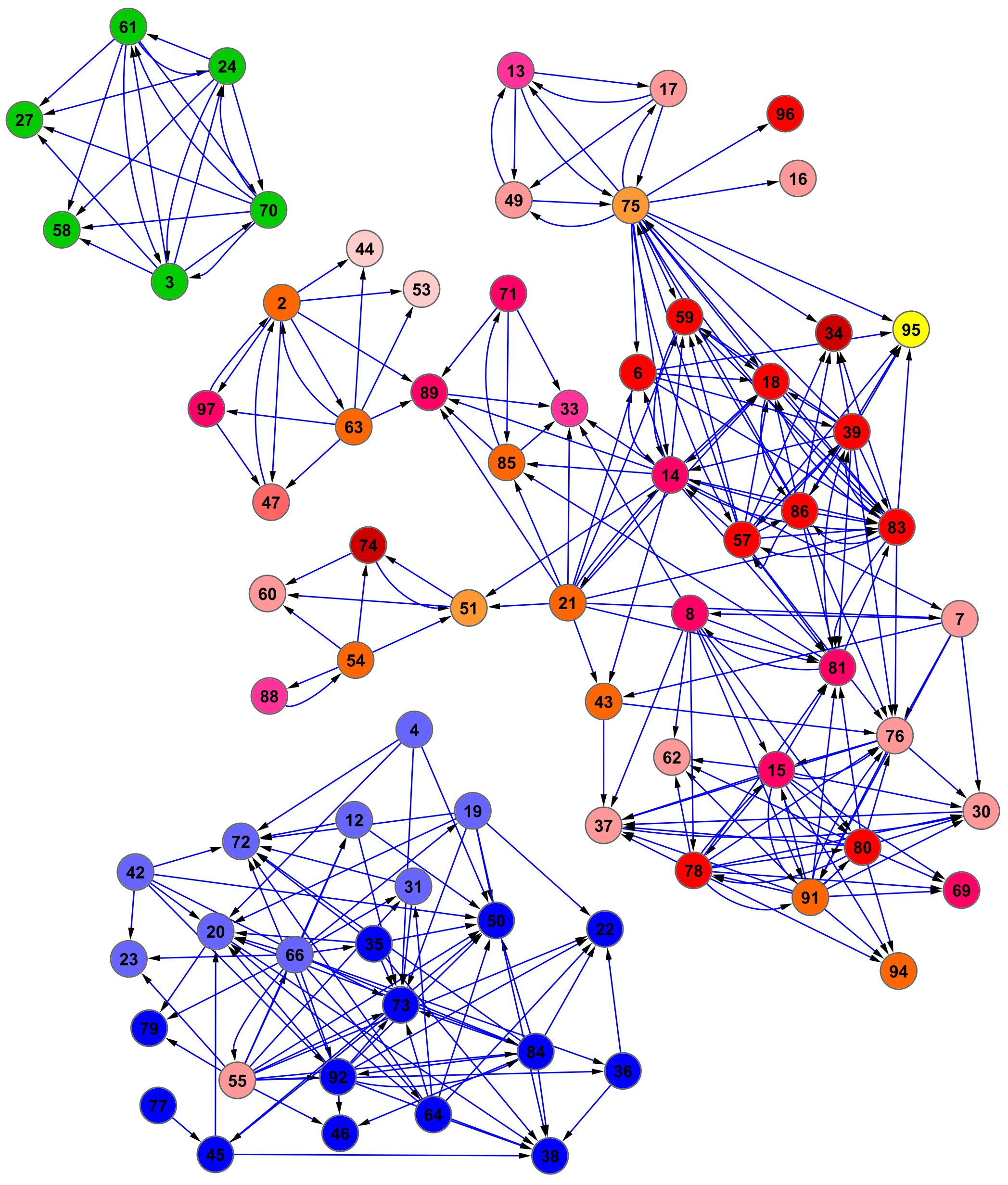


Figure E.9.: Community Graph After 24 Weeks: *Color Information in Table E.19.*

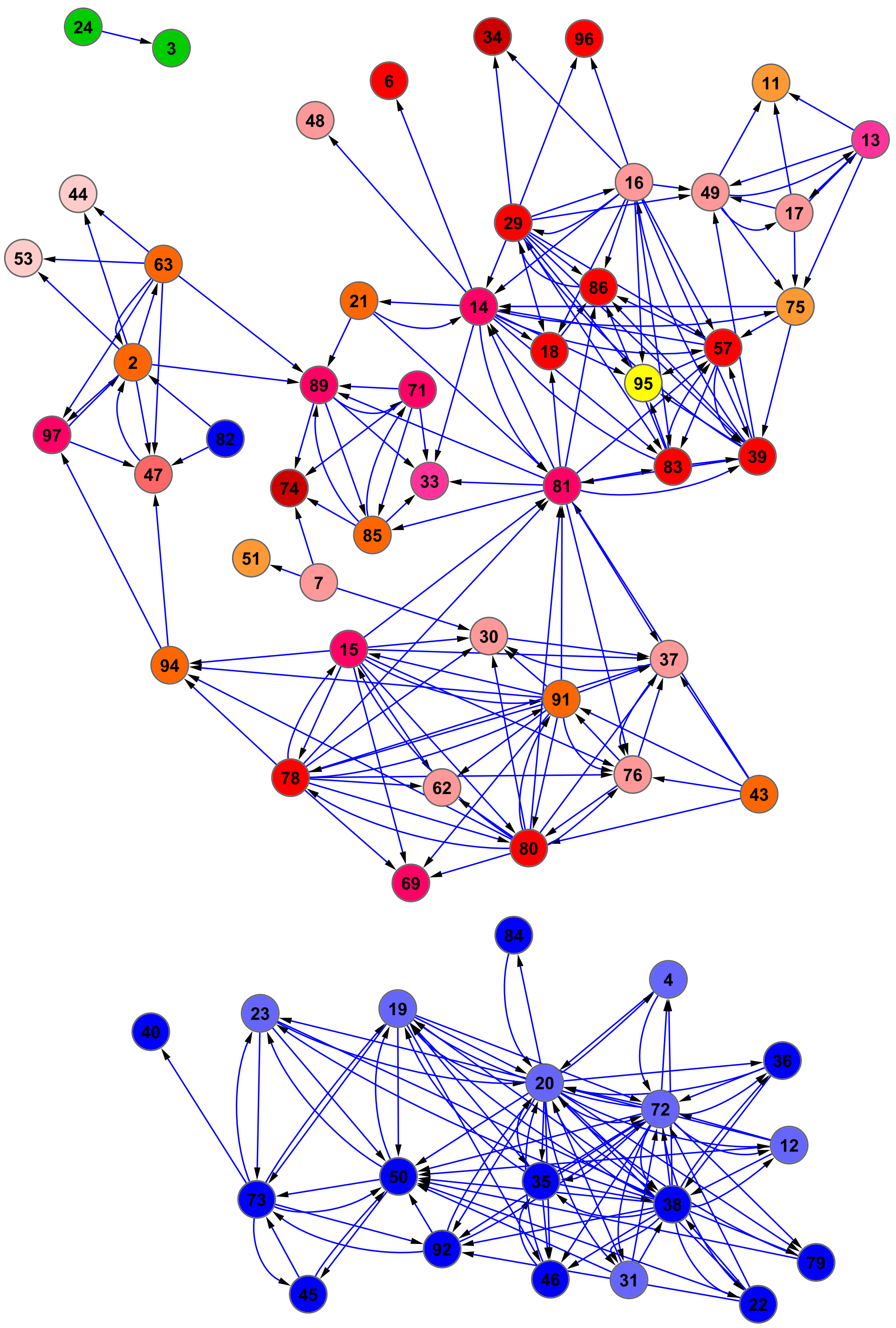


Figure E.10.: Community Graph After 28 Weeks: *Color Information in Table E.19.*

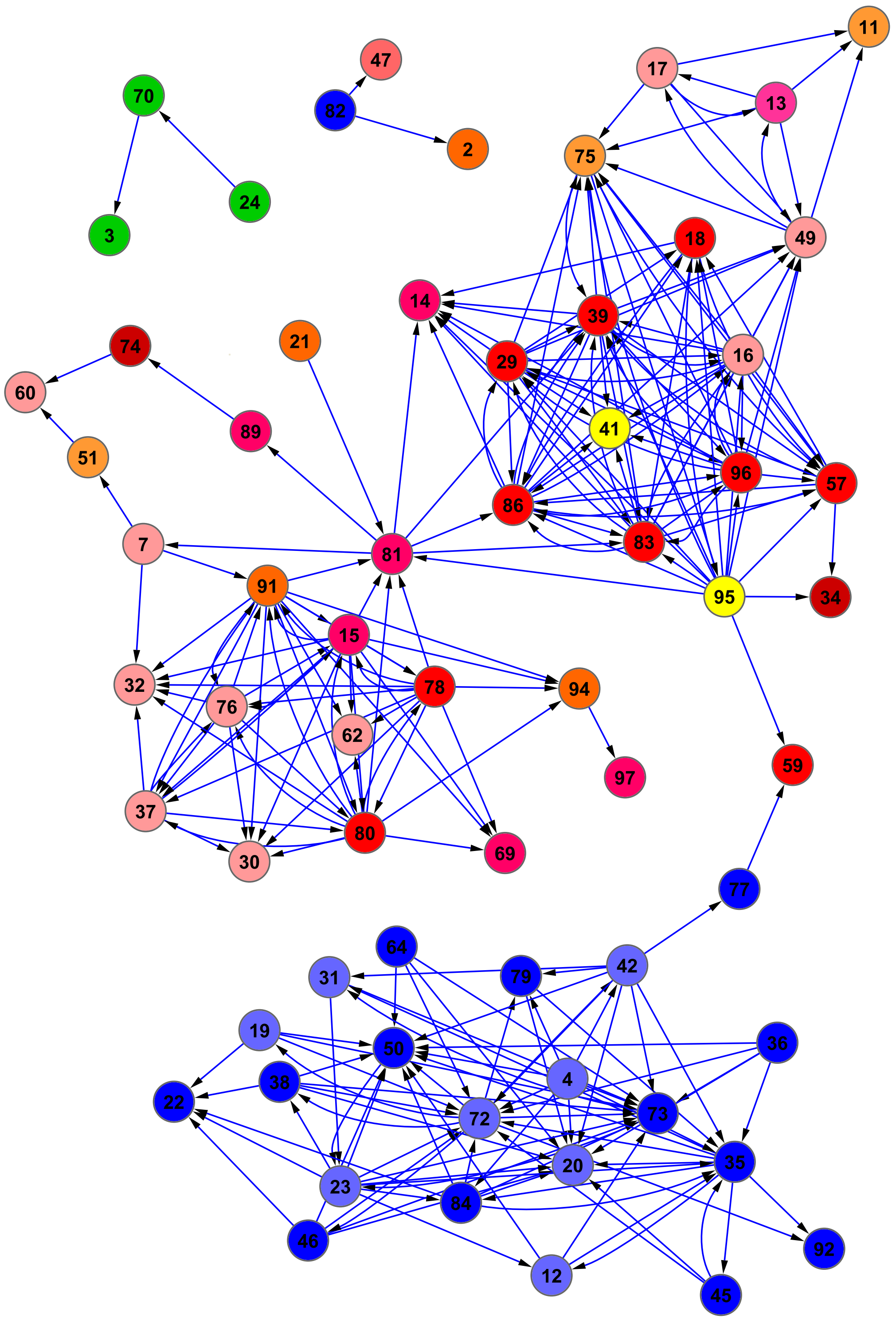


Figure E.11.: Community Graph After 32 Weeks: *Color Information in Table E.19.*

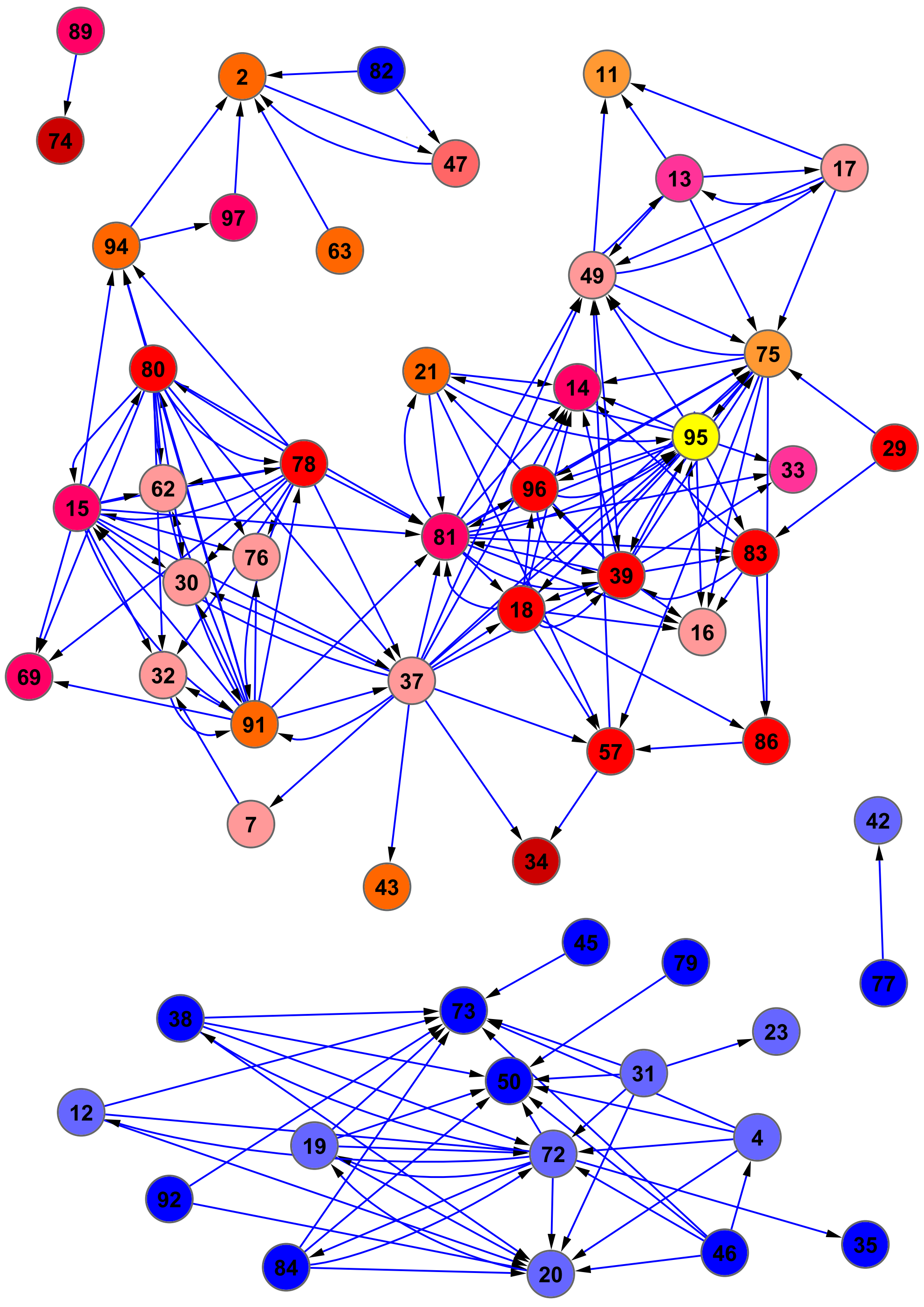


Figure E.12.: Community Graph After 36 Weeks: *Color Information in Table E.19.*

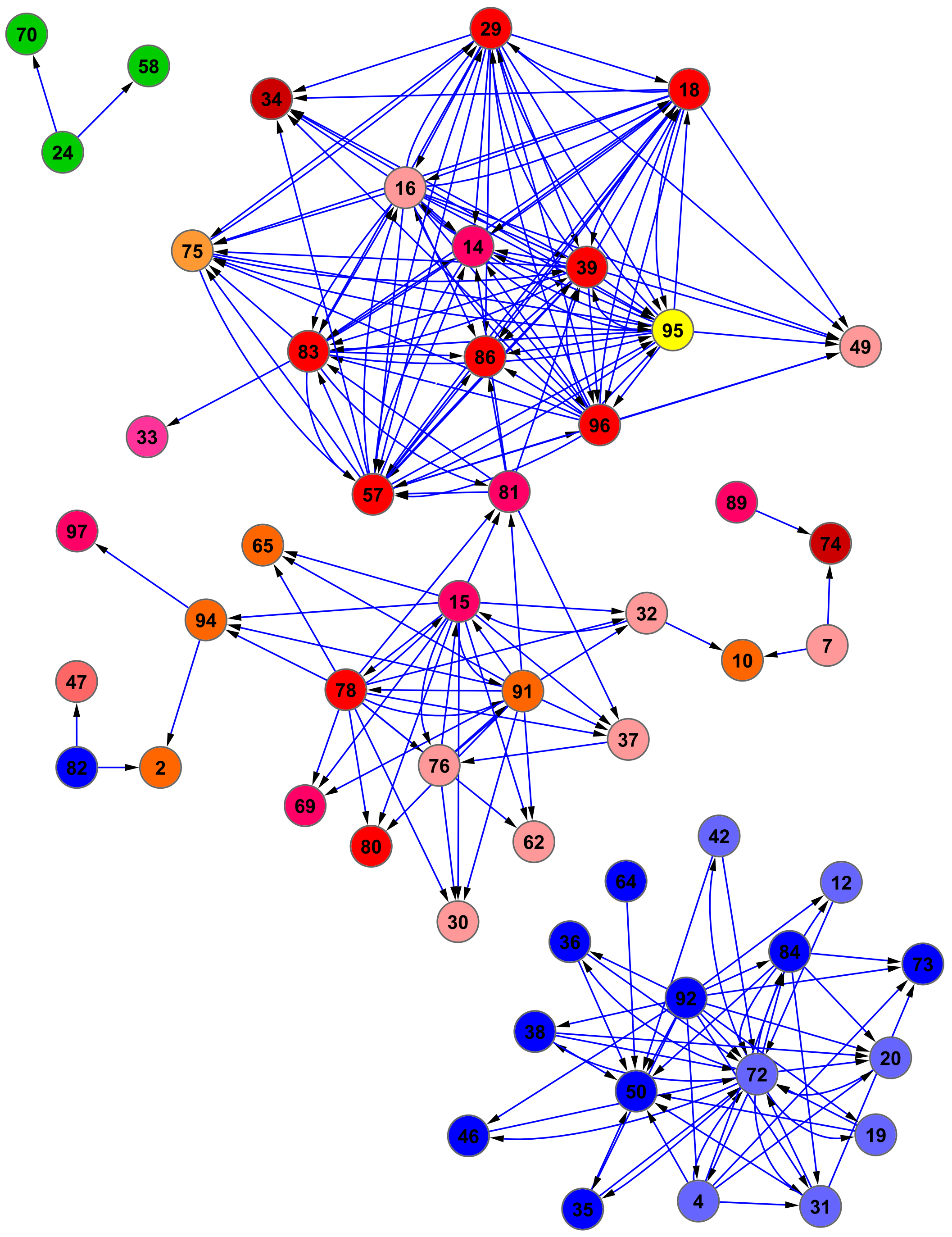


Figure E.13.: Community Graph After 40 Weeks: *Color Information in Table E.19.*

Bibliography

[1] RSS Advisory Board. Really simple syndication specification. Version 2.0.11, March 2009. http://www.rssboard.org/rss-specification.

[2] M. May G. Karlsson, V. Lenders. Delay-tolerant broadcasting. In *Proceedings of the ACM SIGCOMM Workshops (CHANTS)*, 2006.

[3] C. Wacha. Wireless ad hoc podcasting with handhelds. Master's thesis, Swiss Federal Institute of Technology, 2007.

[4] Bram Cohen. The bittorrent protocol specification, February 2008. http://www.bittorrent.org/beps/bep_0003.html.

[5] W. Polk D. Solo R. Housley, W. Ford. Rfc 2459: Internet x.509 public key infrastructure certificate and crl profile, 1999.

[6] J. P. Hubaux S. Capkun, L. Buttyan. Self-organized public-key management for mobile ad-hoc networks. In *IEEE Transactions on Mobile Computing*, 2003.

[7] P. R. Zimmermann. *The Official PGP User's Guide*. MIT press, 1995.

[8] L. Buttyan S. Capkun, J. P. Hubaux. Mobility helps peer-to-peer security. In *IEEE Transactions on Mobile Computing*, 2006.

[9] V. Issarny J. Liu. Enhanced reputation mechanism for mobile ad hoc networks. *In Proceedings of the 2nd International Conference on Trust Management*, 2995:48–62, 2004.

[10] S. Hailes A. Abdul-Rahman. Supporting trust in viral communities. In *Proceedings of the 33rd Hawaii International Conference on System Science*, 2000.

[11] A. Halberstadt L. Mui, M. Mohtsahemi. Ratings in distributed systems: A bayesian approach. In *Proceedings of the 11th Workshop on Information Technologies and Systems*, 2001.

[12] J. Y. Le Boudec S. Buchegger. Performance analysis of the confidant protocol. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc' 02)*, 2002.

[13] J. Y. Le Boudec S. Buchegger. The effect of rumor spreading in reputation systems for mobile ad-hoc networks. In *WiOpt' 03: In Proceedings of the 2003 Conference on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003.

[14] J. Y. Le Boudec S. Buchegger. A robust reputation system for p2p and mobile ad-hoc networks. In *Proceedings of the 2nd Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[15] J. Y. Le Boudec S. Buchegger. Self-policing mobile ad-hoc networks by reputation systems. *IEEE Communications Magazine*, v43:101 – 107, 2005.

[16] L. Capra D. Quercia, S. Hailes. B-trust: Bayesian trust framework for pervasive computing. In *Proceedings of the 4th IEEE International Conference on Trust Management (iTrust)*, 2006.

[17] L. Capra D. Quercia, S. Hailes. Lightweight distributed trust propagation. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining (ICDM' 07)*, 2007.

[18] L. Capra D. Quercia, S. Hailes. Mobirate: Making mobile raters stick to their world. In *UbiComp' 08: Proceedings of the 10th International Conference on Ubiquitous Computing*, 2008.

[19] J. R. Douceur. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[20] B. Levine C. Piro, C. Shields. Detecting the sybil attack in mobile ad hoc networks. In *Securecomm and Workshops*, 2006.

[21] P. B. Gibbons A. Flaxman H. Yu, M. Kaminsky. Sybilguard: Defending against sybil attacks via social networks. In *SIGCOMM ' 06: Proceedings of the 2006 Conference on Applications, Technologies, and Protocols for Computer Communications*, 2006.

[22] L. Capra D. Quercia, S. Hailes. Tata: Towards anonymous trusted authentication. In *Proceedings of the 4th International Conference on Trust Management*, 2006.

[23] P. Resnick E. J. Friedman. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10:173–199, 2001.

[24] S. Y. Chan J. Crowcroft P. Hui, E. Yoneki. Distributed community detection in delay tolerant networks. In *MobiArch' 07: Proceedings of the 2nd ACM / IEEE International Workshop on Mobility in the Evolving Internet Architecture*, 2007.

[25] M. Roe G. O'Shea. Child-proof authentication for mipv6 (cam). *ACM SIGCOMM Computer Communication Review*, 31:4–8, 2001.

[26] C. Castelluccia G. Montenegro. Statistically unique and cryptographically verifiable (sucv) identifiers and addresses. In *NDSS'02*, 2002.

[27] N. Feamster D. G. Andersen, H. Balakrishnan. Accountable internet protocol (aip). In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM '08)*, 2008.

[28] S. Hailes D. Quercia. Mate: Mobility and adaptation with trust and expected-utility. *International Journal of Internet Technology and Secured Transactions (IJITST)*, 2007.

[29] K. Xu S. Y. Chan, P. Hui. Community detection of time-varying mobile social networks. work in progress paper.

[30] B. Boe K. Almeroth B. Y. Zhao G. Swamynathan, C. Wilson. Do social networks improve e-commerce? a study on social marketplaces. In *Proceedings of the first workshop on Online Social Networks (WOSP' 08)*, 2008.

[31] E. Goodman E. Paulos. The familiar stranger: Anxiety, comfort, and play in public places. In *Proceedings of the 4th International Conference on Trust Management*, pages 298–312, 2004.

[32] I. Farkas T. Vicsek G. Palla, I Derenyi. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435, 2005.

[33] V. Lenders M. May G. Karlsson O. R. Helgason, F. Legendre. Performance of opportunistic content distribution.

[34] S. Hailes L. Yan. Cooperative packet relaying model for wireless ad hoc networks. In *Proceeding of the 1st ACM International Workshop on Foundations of Wireless Ad Hoc and Sensor Networking and Computing (FOWANC' 08)*, 2008.

[35] A. Adleman R. Rivest, R. L. Shamir. A method for obtaining digital signatures and public cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[36] Standards for Efficient Cryptography Group (SECG). Sec 1: Elliptic curve cryptography, 2000.

[37] Standards for Efficient Cryptography Group (SECG). Sec 2: Recommended elliptic curve domain parameters, 2000.

[38] E. Tromer A. Shamir. Factoring large numbers with the twirl device. In *Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO)*, 2003.

[39] K. Lauter. The advantages of elliptic curve cryptography for wireless security. *IEEE Wireless Communications*, 11:62–67, 2004.

[40] L. Capra V. Zanardi. Social ranking: Uncovering relevant content using tag-based recommender systems. In *Proceedings of the ACM Conference On Recommender Systems*, pages 51–58, 2008.

[41] V. Zanardi D. Quercia, L. Capra. Selecting trustworthy content using tags. *Invited Paper at SECRYPT, Special Session on Trust in Pervasive Systems and Networks*, 2008.

[42] V. Kalogeraki T. Repantis. Decentralized trust management for ad-hoc peer-to-peer networks. In *Proceedings of the 4th international workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC' 06)*, 2006.