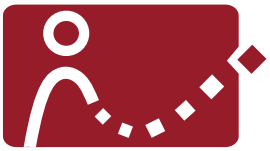
Group Formation for Asymmetric DC-nets



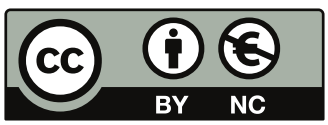
Gruppenzusammenstellung für asymmetrische DC-Netzwerke
Bachelorarbeit von Simon Nicolas Helmut Becker
Tag der Einreichung: 2. Oktober 2018

1. Gutachter: Prof. Dr. Max Mühlhäuser
2. Gutachter: Dr. Tim Grube

Darmstadt, 2. Oktober 2018

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Telekooperation
Prof. Dr. Max Mühlhäuser



Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Name: Simon Nicolas Helmut Becker Datum: Unterschrift:

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Name: Simon Nicolas Helmut Becker Date: Signature:

Abstract

Many-to-many communication is a central component of widespread technologies like social networks and the IoT. For many applications, it is desirable that the anonymity of communication participants is preserved. We focus on the protection of sender anonymity in peer-to-peer networks that allow scalable group communication via publish-subscribe overlays. While DC-nets can be used for achieving sender and receiver anonymity for many-to-many communication, they also induce a considerable communication and computation overhead. We address this challenge by introducing a modified version of DC-nets that induces less overhead than classical DC-nets while preserving their sender anonymity properties. Moreover, we present challenges that arise when combining publish-subscribe overlays with DC-nets. We introduce a novel technique that addresses the presented challenges, hereby protecting the anonymity of senders during its initialization and runtime. As our approach allows for anonymization groups of configurable size, we also analyze how groups that are suitable for this purpose can be formed.

Zusammenfassung

Many-to-Many Kommunikation ist ein zentraler Bestandteil von weitverbreiteten Technologien - beispielsweise von sozialen Netzwerken und dem IoT. Für viele Anwendungen ist es wünschenswert, dass die Anonymität der Kommunikationsteilnehmer geschützt wird. Wir konzentrieren uns auf den Schutz der Sender-Anonymität in Peer-to-Peer Netzwerken, in welchen Publish-Subscribe Overlays eine skalierbare Gruppenkommunikation ermöglichen. Während DC-Netzwerke genutzt werden können, um Sender- und Empfänger-Anonymität für Many-to-Many Kommunikation zu erreichen, verursachen sie auch einen erheblichen kommunikativen und rechnerischen Mehraufwand. Wir reagieren auf diese Herausforderung mit der Einführung einer modifizierten Variante von DC-Netzwerken. Diese erzeugt einen geringeren Mehraufwand als klassische DC-Netzwerke und behält gleichzeitig den Schutz der Sender-Anonymität bei. Außerdem präsentieren wir Herausforderungen, die auftreten, wenn Publish-Subscribe Overlays mit DC-Netzwerken verbunden werden. Wir stellen eine neuartige Methode vor, die diese Herausforderungen löst und somit die Anonymität von Sendern während ihrer Initialisierung und Laufzeit schützt. Da unser Ansatz es ermöglicht, Gruppen konfigurierbarer Größe zur Anonymisierung zu nutzen, analysieren wir auch, wie Gruppen gebildet werden können, die für diese Aufgabe geeignet sind.

Contents

1 Introduction

Nowadays, more aspects of our daily life are connected to digital applications than ever before. While, some decades ago, digital communication has been very costly, advances in network technology made it affordable for the majority of the general population. With an increasing number of people that have access to the Internet, more and more opportunities for so called many-to-many communication arise. In contrast to traditional one-to-one communication, this communication model allows multiple senders and multiple receivers to interact with each other. Hence, it is a central aspect of a variety of popular technologies, including social networks like Twitter and Facebook, the Internet of Things (IoT) and "Smart Technologies" like smart power grids and smart homes. With these technologies being part of our daily life, many-to-many communication is already very relevant and will further increase in importance. However, privacy is often forgotten despite its importance for the communication participants.

Nowadays, it is known that global governmental mass surveillance programs exist and pose a constant threat to the privacy of the general population. While the existence of such programs was already suspected, it took many years for actual information to be available to the public. For instance, the ECHELON [10] mass surveillance system has already been used for more than two decades before its existence was confirmed by official documents in the early 2000s. Yet, it took until 2013 for the extent of western surveillance programs to come to light. The classified documents leaked by Edward Snowden revealed a multitude [35] of global mass surveillance programs invading the privacy of the general population. Stoycheff [45] indicated that mass surveillance has a psychological effect on individuals and can discourage them from voicing controversial opinions. This is in direct opposition to the principle of freedom of speech. Furthermore, the European Court of Human Rights ruled that the mass surveillance conducted by GCHQ violated human rights [7]. However, governmental surveillance is not the only reason for proper privacy protection as private information can easily fall into the wrong hands. For instance, leaks of poorly secured information [39, 50, 20] and the sharing of private information with third parties [9, 43] can have a severe impact on user privacy.

For demonstrating the importance of privacy protection, a basic definition of privacy is necessary. We refrain from defining privacy due to the complexity of the topic and instead refer to the following definition: *Privacy is the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others* [49]. To illustrate the importance of privacy, we now assume its complete absence. It has to be assumed that everybody knows everything about *someone* and that everybody might use this someones personal data in any way they want to. This is critical for multiple reasons. For one, personal information can be used to influence and manipulate people into doing things that they would otherwise not do. A prime example for this is targeted advertising [27] that heavily relies on personal information. Furthermore, privacy becomes an important factor in the context of oppressive governments. It enables people to give a voice to their opinions and beliefs without having to fear political or religious persecution or persecution regarding other factors of their identity. Hence, a world with the complete absence of privacy would be a dystopian one. Fortunately we do not live in such a world. Still, there is often a *lack of privacy* that can be more or less invasive.

We focus on privacy in the ubiquitous context of digital communication. The encryption of messages can protect the privacy of communication participants to some extent. However, the content of messages is not the only aspect worthy of protection. Even if the actual content of a message is not known to an adversary, it can still detect that certain participants communicate with each other. This is very critical as it can have a direct impact on the privacy of the communication participants. For instance, oppressive governments may use such information to detect potential supporters of political adversaries and persecute them. Therefore, receiver and sender anonymity are very relevant for the privacy of users in digital communication.

We focus on the aspect of sender anonymity in peer-to-peer (P2P) networks as techniques for reaching a sufficient level of receiver-anonymity in this setting have already been presented [41, 48]. Furthermore, we focus on many-to-many communication. Past research resulted in promising techniques for protecting sender anonymity in this setting, for instance, Dining Cryptographers networks (DC-nets). However, they are problematic with regards to both their communicational and computational overhead due to the extensive use of cover traffic (CT). This overhead directly impacts the performance and scalability of the networks in which such techniques are employed. As scalability is a crucial aspect for many-to-many communication, we introduce Asymmetric Dining Cryptographers networks (ADC-nets) as modified version of DC-nets with reduced overhead and comparable sender protection.

The structure of this thesis is as follows: Required background and definitions are summarized in Chapter 2 and related work referred to in Chapter 3. Our theoretical approach is introduced in Chapter 4 and the respective prototypical implementation presented in Chapter 5. Afterwards, both are evaluated in Chapter 6. Chapter 7 concludes this thesis.

2 Background

In this chapter, relevant background information is introduced: First, we discuss different types of network addressing in Section 2.1. Afterwards, we introduce a terminology regarding anonymity in communication in Section 2.2. It is followed by the presentation of the adversary model which is used throughout this thesis in Section 2.3. Then, the CT anonymization technique is presented in Section 2.4. With Mix networks (Mix nets), a technique for the anonymization of one-to-one communication is described in Section 2.5. As Mix nets are not suitable for many-to-many communication, we rely on DC-nets that are presented in Section 2.6. In Section 2.7, the concept of group formation is introduced. Afterwards, the aspects of overlays that are most relevant for this thesis are summarized in Section 2.8. Due to its importance for leader election, we refer to the byzantine generals problem in Section 2.9. Then, the topic of leader elections in distributed systems is presented in Section 2.10. Final, we present our assumptions regarding the underlying P2P network which is used throughout this thesis in Section 2.11.

2.1 Network Addressing

While more techniques for network addressing exist, we focus on the main classes used for this task:

- Most of the global network traffic is one-to-one communication and uses *unicast* addressing. While not designed for this purpose, unicast can also be used for one-to-many communication by sending the same message multiple times, once for each receiver. However, this results in additional overhead for both the sender and the network and is therefore impractical.
- *Multicast* in turn allows to address a group of peers and is therefore predestined for one-to-many communication. The sender only has to send each message once and messages are efficiently delivered to all recipients by the network.
- While it was commonly used before multicast was widely available, *broadcast* is nowadays a special case of multicast. It allows the addressing of all peers in a network and is therefore designated for one-to-all communication. Furthermore, similar to multicast, the network takes care of an efficient delivery of broadcast messages.

Many multicast and broadcast addressing implementations are not ensuring a reliable transport of messages. However, various techniques that provide reliable multicast [19, 38, 8] and reliable broadcast [11, 28] have been presented in past research. These techniques can be used if the reliable transport of messages is crucial.

2.2 Anonymous Communication

Anonymity in the context of communication is a complex concept. We therefore rely on the extensive terminology presented by Pfitzmann et al. [40] and summarize the aspects that are

most relevant for this thesis. Introductory, we refer to the following definition: *Anonymity of a subject from an attacker's perspective means that the attacker cannot sufficiently identify the subject within a set of subjects, the anonymity set* [40].

Anonymity sets are useful for measuring the anonymity of subjects. In general, larger anonymity sets make it harder for adversaries to identify a specific subject, hence resulting in stronger anonymity. However, multiple subjects within an anonymity set can collude by exchanging information regarding their actions. Colluding subjects can use this information for reducing the effective anonymity set size. This makes it easier for them to identify a non-colluding subject within the anonymity set.

As *Unlinkability* is an important aspect of anonymity in a communication context, we also refer to the following definition: *Unlinkability of two or more items of interest (IOIs, e.g., subjects, messages, actions, ...) from an attacker's perspective means that within the system (comprising these and possibly other items), the attacker cannot sufficiently distinguish whether these IOIs are related or not* [40].

By definition, the subjects of an anonymity set are indistinguishable from each other from the view of an adversary. Therefore, communication technologies in which messages contain sender and receiver addresses, prevent both sender and receiver anonymity. To achieve anonymity of senders and receivers, messages have to be decoupled from sending and receiving subjects. Only this way, senders and receivers cannot be linked to messages.

Anonymization techniques like Mix nets and DC-nets decouple the participants of a communication from messages and hence allow unlinkability. These techniques are presented in the Sections 2.5 and 2.6.

2.3 Adversary Model

To evaluate the security aspects of existing techniques and our new approach, we formulate the following assumptions regarding the capabilities of an adversary:

1. *Global*: has knowledge of all communication in the network.
2. *Eavesdropping*: can eavesdrop on every observed message.
3. *Insider*: can be part of the network and take part in the communication.
4. *Colluding*: can collude with other adversaries.
5. *Non-disruptive*: does not intentionally disrupt the communication.

We assume global and eavesdropping adversaries as this allows us to apply our approach to networks that are monitored by adversaries. Such adversaries can read every message sent over the network. Furthermore, they can detect the peer at which a certain message was first forwarded. As this peer is the sender of the message, global and eavesdropping adversaries pose a serious threat to sender anonymity. Also, we assume that insider adversaries exist and can take part in the communication. This allows the application of our approach to P2P networks with untrusted peers. Furthermore, we assume that adversaries can cooperate with each other in situations where colluding allows them to gain an advantage over other peers. While the first four assumptions present a realistic adversary model for P2P networks, the assumption of adversaries being non-disruptive is problematic. In a real-world network, adversaries may

willingly disrupt the communication between peers, for instance, by dropping messages and inducing congestion. However, we do not focus on the aspect of disruption-prevention and hence expect adversaries to be non-disruptive. Yet, we present challenges regarding disrupting adversaries throughout this thesis and refer to existing techniques that address these challenges.

2.4 Cover Traffic

CT is an anonymization technique for reducing the linkability of senders and messages by adding random noise to the communication. This can cover otherwise distinctive traffic patterns, thus reducing the amount of information that an adversary can gain from observing a communication. Many techniques that can be used for achieving sender anonymity, e.g., Mix nets and DC-nets, rely on CT as central component. However, CT often induces an enormous overhead that heavily impacts the scalability of such technologies.

We now refer to past research in the field of CT:

- Levine et al. [34] analyzed the effectiveness of CT used in Mix nets as protective measure against timing analysis attacks. They showed that certain timing analysis attacks can be very effective against conventional CT. Furthermore, they proposed a new CT technique that is more resilient to timing attacks.

- Mallesh et al. [37] proposed receiver-bound CT as technique for preventing statistical disclosure attacks. It relies on Mixes sending CT that mimics the sending patterns of actual users. They showed that their technique can be beneficial with regards to sender anonymity.

- Grube et al. [26] analyzed how the overhead caused by CT can be reduced. They evaluated the impact of different parameters on CT efficiency and anonymity. Furthermore, they presented a technique that can drastically improve efficiency while maintaining a reasonable level of anonymity. However, they also showed that this technique is susceptible to intersection attacks that can be used by adversaries to reduce sender anonymity after several messages have been emitted by the sender.

2.5 Mix Networks

Chaum [13] introduced the concept of Mixes and Mix nets that can be used for hiding the identity of communication participants. Mix nets contain a number of Mixes that act as intermediates between a number of peers communicating with each other.

A message sent over a series of Mixes is encrypted by the sender using multiple layers of asymmetric cryptography to reduce the traceability of the message. For each Mix in the series, a corresponding encryption layer is added to the message. When the message passes a Mix, the corresponding encryption layer is removed from the message. In addition to the layered encryption of messages, each Mix performs a number of steps for preventing the linking of a message to both sender and receiver. Therefore, the actual receiver of a message can only link it to the last Mix in the series but not to its actual sender. Furthermore, no Mix in the series knows both the sender and the recipient of a message. Hence, even untrusted entities can be used as Mixes. However, it has to be prevented that all Mixes in the series are controlled by the same

adversary. Such an adversary has the capabilities to link each message sent over the series to its sender and its receiver.

There are several applications that are based on the concept of Mix nets, for instance Tor [46] and Java Anon Proxy [6] which is also known as JonDo. However, many of them are not real Mix nets as they skip specific anonymization tasks in order to improve performance. While this allows real-time communication, it also makes it easier for adversaries to attack the anonymity of users of such applications. For instance, Bauer et al. [4] showed that optimizations applied to Tor made it vulnerable to low-resource traffic analysis attacks.

Mix nets were designed for one-to-one communication. Their use in many-to-many communication is problematic as a new Mix route with new asymmetric keys has to be established for each sender-receiver pair of the communication. In addition to the initialization overhead, the sending of a message to multiple receivers is problematic. Each message has to be duplicated at the sender to be encrypted and sent individually to each recipient over the individually established Mix routes. This results in a computational and communication overhead that drastically increases with the size of the communication group. Hence, using Mix nets for many-to-many communication is infeasible.

2.6 Dining Cryptographers Networks

DC-nets were first introduced by Chaum [12] and can be used for preserving sender and receiver anonymity in a many-to-many setting. Communication within DC-nets is split into rounds in which each participant publishes a message by broadcasting it to the network. The contents of these messages are protected against eavesdropping adversaries by an encryption scheme that is explained in Section 2.6.2. Furthermore, the length of all messages is the same and previously agreed upon. Hereby, an anonymity set comprising all participants of the DC-net is maintained.

The encryption scheme used by DC-nets relies on cryptographic secrets that are established pair-wise among the DC-net participants. We present the establishment of these secrets as part of an initialization phase in Section 2.6.1. Afterwards, we describe the communication model of DC-nets in Section 2.6.2. In Section 2.6.3, we discuss the security of DC-nets. Final, we present a number of challenges that are associated with DC-nets in Section 2.6.4.

2.6.1 Initialization

Each DC-net participant requires a pair-wise secret for each other participant in the DC-net for each round of communication. As each secret is only used for one round, we refer to them as *round secrets*. Furthermore, the length of each round secret has to be equal to the message length used for DC-net communication due to the way they are used.

While it is possible to establish each round secret with an own key agreement, it is more convenient to establish one *master secret* per pair of DC-nets participants and use it for the generation of round secrets. Hence, pair-wise master secrets among the DC-net participants are established in an initial key agreement phase. After this phase, every participant shares a master secret with each other participant in the network.

Example 2.6.1: Master Secret Establishment

Given a DC-net with a participant set $P = \{A, B, C\}$ and a secure key agreement protocol (e.g., Diffie-Hellman key exchange (DH) [44]) that produces a master secret ms_{XY} for the participants $X, Y \in P$. For simplicity, assume that the function $ka(X, Y) = ms_{XY}$ represents the key agreement between the participants X and Y in which they agree on ms_{XY}.

Scenario: A, B and C establish their master secrets.

$$\text{A} : ms_{AB} = ka(A, B) \text{ and } ms_{AC} = ka(A, C)$$

$$\text{B} : ms_{AB} = ka(A, B) \text{ and } ms_{BC} = ka(B, C)$$

$$\text{C} : ms_{AC} = ka(A, C) \text{ and } ms_{BC} = ka(B, C)$$

Traditionally, each master secret is split into a number of round secrets. Hereby, only a *finite* number of round secrets can be obtained from each master secret. Therefore, this approach is problematic with regards to usability as the communication is limited to a finite number of rounds. However, it is also possible to use the master secrets as seeds for cryptographically secure pseudo-random number generators (CSPRNGs). Hereby, an arbitrary number of round secrets can be dynamically generated. However, as shown in Section 2.6.3, this is not as secure as the traditional approach.

Example 2.6.2: Round Secret Derivation

Given a DC-net with the participants A, B and C, the master secrets $ms_{AB} = 11000100, ms_{AC} = 00101010$ and $ms_{BC} = 01010100$. Assume a CSPRNG CS and a message length of 4 to be publicly known. Furthermore, assume $rs_{XY,r}$ as round secret for round r that was derived using the master secret ms_{XY}. For simplicity, assume that CS can be used as $CS(ms_{XY}, r) = rs_{XY,r}$ and returns the first four digits of ms_{XY} if r is even and the last four digits of ms_{XY} otherwise.

Scenario 1: A, B and C compute as many round secrets as possible using the traditional approach:

Participants	Round 1	Round 2
A and B	$rs_{AB,1} = 1100$	$rs_{AB,2} = 0100$
A and C	$rs_{AC,1} = 0010$	$rs_{AC,2} = 1010$
B and C	$rs_{BC,1} = 0101$	$rs_{BC,2} = 0100$

More round secrets cannot be computed.

Scenario 2: A, B and C compute the round secrets for the first four rounds using the CSPRNG CS:

Participants	Round 1	Round 2
A and B	$rs_{AB,1} = CS(ms_{AB}, 1) = 0100$	$rs_{AB,2} = CS(ms_{AB}, 2) = 1100$
A and C	$rs_{AC,1} = CS(ms_{AC}, 1) = 1010$	$rs_{AC,2} = CS(ms_{AC}, 2) = 0010$
B and C	$rs_{BC,1} = CS(ms_{BC}, 1) = 0100$	$rs_{BC,2} = CS(ms_{BC}, 2) = 0101$

Participants	Round 3	Round 4
A and B	$rs_{AB,3} = CS(ms_{AB}, 3) = 0100$	$rs_{AB,4} = CS(ms_{AB}, 4) = 1100$
A and C	$rs_{AC,3} = CS(ms_{AC}, 3) = 1010$	$rs_{AC,4} = CS(ms_{AC}, 4) = 0010$
B and C	$rs_{BC,3} = CS(ms_{BC}, 3) = 0100$	$rs_{BC,4} = CS(ms_{BC}, 4) = 0101$

An arbitrary number of round secrets can be computed.

2.6.2 Communication

As already mentioned in the introduction, the communication within DC-nets is split into rounds in which each participant publishes a message by broadcasting it to the network. As eavesdropping adversaries can link these messages to their senders, their content is encrypted using the DC-net encryption scheme. We refer to the encrypted messages as *DC-net messages* and to the content that is encrypted as *secret messages*.

The DC-net encryption scheme allows the distribution of one secret message per round. This is realized as follows: Each participant creates a DC-net message by XORing a secret message with all the round secrets of this participant for the current round. The secret message can either be empty or contain actual content that the participant wants to publish via the DC-net. DC-net messages constructed from empty secret messages hereby function as CT.

Assuming that only one participant published a non-empty secret message, this secret message can be extracted by XORing all DC-net messages of the round. The challenge of multiple non-empty secret messages being published in the same round is addressed in Section 2.6.4.

Example 2.6.3: DC-net En-/Decryption

Given a DC-net with the participants A, B and C with $rs_{AB} = 10001011$, $rs_{AC} = 10111001$ and $rs_{BC} = 00110101$ as 8-bit round secrets for the current round.

Scenario 1: A wants to distribute a secret message $m_A = 11111000$ and both B and C send empty secret messages.

The following DC-net messages are computed and broadcasted:

$$c_A = m_A \oplus rs_{AB} \oplus rs_{AC} = 11001010$$

$$c_B = rs_{AB} \oplus rs_{BC} = 10111110$$

$$c_C = rs_{AC} \oplus rs_{BC} = 10001100$$

Every participant can then decrypt the secret message published by A:

$$c_A \oplus c_B \oplus c_C = 11111000 = m_A$$

Scenario 2: A, B and C send empty DC-net messages.

The following DC-net messages are computed and broadcasted:

$$c_A = rs_{AB} \oplus rs_{AC} = 00110010$$

$$c_B = rs_{AB} \oplus rs_{BC} = 10111110$$

$$c_C = rs_{AC} \oplus rs_{BC} = 10001100$$

Every participants can then XOR the received DC-net messages and see that only an empty message was published:

$$c_A \oplus c_B \oplus c_C = 00000000$$

2.6.3 Security

As new round secrets are used for each round, each DC-net message is effectively a one-time pad (OTP) [44]. As OTPs exhibit perfect security, it is impossible for adversaries to decrypt them by using cryptanalysis without background information regarding the used round secret. With the traditional approach for the generation of round secrets, the randomly generated master secrets are split into multiple round secrets. Therefore, it is impossible for adversaries to decrypt DC-net messages unless they have further knowledge regarding the master secrets or the round secrets for specific rounds. However, when using the master secrets as seeds for a CSPRNG, the round secrets are only random to a certain extent and the exchanged DC-net messages therefore not perfectly secure. Instead, the security depends on the security of the used CSPRNG.

While still not perfectly secure, reestablishing the master secrets in regular intervals and using them as new seeds for the CSPRNG might increase the complexity of DC-net message cryptanalysis. However, even if perfectly secure against outsiders, insider adversaries pose a threat to the security of DC-nets as they can

- reduce the effective size of the DC-net anonymity set by colluding with other insiders,
- impersonate honest participants if they control all the communication links over which honest participants can communicate with the rest of the DC-net and
- fake a larger anonymity set size by pretending to be connected to participants that do not actually exist.

2.6.4 Challenges

While DC-nets exhibit overall desirable security properties, a number of challenges exist.

Overhead

DC-nets introduce a massive communicational overhead due to the extensive use of CT. While every participant has to send a DC-net message, only one secret message can be exchanged per round. Furthermore, several XOR operations have to be executed for each DC-net round, resulting in a significant computational overhead. In Section 6.2.2, we compare the overhead of DC-nets to ADC-nets, which we introduce in Section 4.4.

Collisions

Due to missing medium access control (MAC), multiple non-empty secret messages may collide in DC-nets with multiple sending participants. Every participant is allowed to send a non-empty secret message in any round and without any coordination. Therefore, it is possible that multiple participants send non-empty secret messages in the same round, resulting in a collision of those secret messages. This is problematic as most collisions result in a malformed combination of the non-empty secret messages. Only in few cases, one of the secret messages survives a collision.

Example 2.6.4: DC-net Collisions

Given a DC-net with the participants A, B and C with $rs_{AB} = 10001011$, $rs_{AC} = 10111001$ and $rs_{BC} = 00110101$ as 8-bit round secrets for the current round.

Scenario 1: A wants to distribute the secret message $m_A = 11111000$, B wants to distribute the secret message $m_B = 01000100$ and C sends an empty secret message.

The following DC-net messages are computed and sent via broadcast:

$$c_A = m_A \oplus rs_{AB} \oplus rs_{AC} = 11001010$$

$$c_B = m_B \oplus rs_{AB} \oplus rs_{BC} = 11111010$$

$$c_C = rs_{AC} \oplus rs_{BC} = 10001100$$

The secret messages sent by A and B resulted in a collision:

$$c_A \oplus c_B \oplus c_C = 10111100 = m_A \oplus m_B$$

Scenario 2: A wants to distribute the secret message $m_A = 11111000$ and both B and C want to distribute the secret message $m_B = m_C = 01000100$.

The following DC-net messages are computed and sent via broadcast:

$$c_A = m_A \oplus rs_{AB} \oplus rs_{AC} = 11001010$$

$$c_B = m_B \oplus rs_{AB} \oplus rs_{BC} = 11111010$$

$$c_C = m_C \oplus rs_{AC} \oplus rs_{BC} = 11001000$$

The secret messages sent by B and C cancel each other out. Hence, the secret message sent by A survived the collision:

$$c_A \oplus c_B \oplus c_C = 11111000 = m_A \oplus m_B \oplus m_C = m_A$$

While we do not focus on collision detection, we implemented a straightforward collision handling technique for the ADC-net communication of our framework implementation presented in Chapter 5. This technique is presented in Section 5.3.7 and evaluated in Section 6.5. Furthermore, we refer to Section 3.2 for protocols that use slot reservation techniques to prevent the occurrence of collisions.

Disruption

DC-nets are prone to both unintended and planned disruptions. Insider adversaries can easily disrupt the communication of a DC-net by

- omitting DC-net messages and
- provoking collisions by constantly sending non-empty secret messages.

Example 2.6.5: DC-net Disruption

Given a DC-net with the participants A, B and C with $rs_{AB} = 10001011$, $rs_{AC} = 10111001$ and $rs_{BC} = 00110101$ as 8-bit round secrets for the current round. Assume that C is an insider adversary whose goal is the disruption of the communication.

Scenario 1: A wants to distribute the secret message $m_A = 11111000$, B sends an empty secret message and C does not send a message.

The following DC-net messages are computed and sent via broadcast:

$$c_A = m_A \oplus rs_{AB} \oplus rs_{AC} = 11001010$$

$$c_B = rs_{AB} \oplus rs_{BC} = 10111110$$

The secret message sent by A cannot be restored because C refused to send an empty secret message:

$$c_A \oplus c_B = 01110100 \neq m_A$$

Scenario 2: A wants to distribute the secret message $m_A = 11111000$, B sends an empty secret message and C sends a random secret message $m_C = 01111000$.

The following DC-net messages are computed and sent via broadcast:

$$c_A = m_A \oplus rs_{AB} \oplus rs_{AC} = 11001010$$

$$c_B = rs_{AB} \oplus rs_{BC} = 11111010$$

$$c_C = m_C \oplus rs_{AC} \oplus rs_{BC} = 11110100$$

The secret message sent by A cannot be restored because C provoked a collision:

$$c_A \oplus c_B \oplus c_C = 11000100 \neq m_A$$

The disruption of the communication by insider adversaries is critical. However, we do not address this challenge in this thesis and instead refer to past research regarding disruption-prevention in Section 3.1.

2.7 Group Formation

Groups are a ubiquitous concept that does not only apply to network theory. Humans both wittingly or unwittingly form groups for all kinds of reasons, be it for sharing means of transportation or engaging in shared hobbies. Groups in networks are also created for specific purposes and are therefore based on different aspects. Mobile ad-hoc networks [36] for instance, can be seen as groups that are formed based on the locality of nearby mobile devices. While many more techniques for forming groups exist, we focus on flooding and random walks, two approaches that are applicable in P2P networks. Furthermore, we only summarize these approaches shortly and refer to Gkantsidis et al. [22] for a more detailed presentation and comparison of both approaches:

- *Flooding* can be used for sending group participation requests to all peers in a network. While it is very robust, it can easily lead to congestion in the network due to the high number of messages being sent.
- *Random Walks* randomly traverse a graph and can be used for a more coordinated distribution of group participation requests. While random walks generally incur less communication overhead than flooding, they also take longer.

2.8 Overlays

Overlays are “networks” build up on top of other network layers, e.g., the IP layer. They connect a subgroup of the members of underlying layers on a new abstraction layer. Addressing of the overlay members is often done on this new abstraction layer. Furthermore, communication links

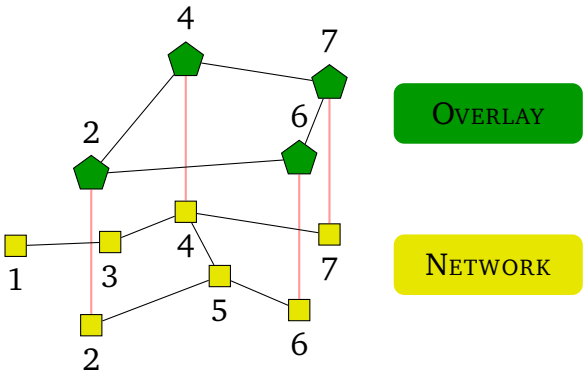


Figure 2.1.: Overlay Example

between the members of an overlay may differ from those of the underlying layer. An exemplary overlay is shown in Figure 2.1.

As overlays have been analyzed thoroughly in past research, we refrain from describing them in detail and rather focus on the aspects that are most relevant for this thesis. Interest-related overlays are described in Section 2.8.1. In Section 2.8.2, the aspect of overlay establishment is addressed.

2.8.1 Interest-related Overlays

Interest-related overlays have a certain topic assigned to them in which all participants of the overlay are interested in. They are based on the publish-subscribe (pub-sub) [17] scheme and can be seen as communication groups in which messages regarding a certain interest are exchanged. For this thesis, we assume that every peer in a network can at any time

- initiate a new overlay,
- join any existing overlay and
- leave any overlay it is part of.

Furthermore, we refer to interest-related overlays as overlays throughout this thesis.

2.8.2 Overlay Establishment

The common goal of overlay establishment techniques is the determination of an efficient distribution tree that connects a specific group of peers and therefore allows for message distribution on the overlay. What technique is most suitable for overlay establishment heavily depends on the purpose of the established overlay. While we do not focus on overlay establishment, we briefly discuss existing privacy-friendly overlay establishment techniques in Section 3.3.

2.9 Byzantine Generals Problem

The Byzantine Generals Problem [32] describes the challenge of reaching multi-party agreements when parties can only communicate with each other over unreliable channels. The

complexity of this challenge is further increased by the existence of malicious entities that try to manipulate this process, for instance, by sending conflicting information to different parties. While different techniques can be used to address the problem, we do not describe them in detail and instead refer to the use of signatures as described by Lamport et al. [32]. By using this approach, it can be ensured that no party can send conflicting information to different parties without being detected.

2.10 Leader Election

Distributed systems often require entities that assume a leading role, for instance, for the coordination of tasks. The field of leader election addresses the challenge of selecting such an entity from a set of candidates. Due to the thorough research in the field, a multitude of leader election algorithms exists. We focus on election algorithms that are conducted in a distributed manner and elect exactly one leader. For instance, the Bully Election Algorithm [21] can be used for electing a single coordinating entity in a distributed system.

Many leader election algorithms function comparable to political elections where a leader is elected from a set of candidates based on the number of votes casted for each candidate. Yet, in some scenarios it is desirable that a leader is randomly chosen from a set of candidates. While this poses no challenge for distributed systems with a trusted entity that can randomly select a leader from the set of candidates, it is a difficult task for systems without such a trusted entity. Traditional leader elections algorithms, which elect a leader based on the number of votes, allow colluding adversarial candidates to cast their votes for one of their peers, hereby effectively manipulating the election.

Example 2.10.1: Traditional Leader Election

Given a set $C = \{1, 2, 3, 4, 5, 6, 7\}$ of candidates that take part in an election. Assume, that non-colluding candidates randomly select one of the candidates and that the candidate with the most votes is elected as leader. Furthermore, assume that the candidates 1, 2 and 3 are colluding.

Scenario: Traditional Election

Candidate	1	2	3	4	5	6	7	
Voted for	3	3	3	4	7	4	6	⇒ Elected Leader: 3

Distributed elections are effectively multi-party agreements. Therefore, the byzantine generals problem presented in Section 2.9 applies and has to be addressed. We focus on election schemes that prevent both non-colluding and colluding adversarial candidates from gaining an unfair advantage and refer to these schemes as *robust*.

Ben-Or et al. [5] introduced the *collective coin-flipping problem* as potential way of using coin-flipping for robust leader election. Based on this problem, Feige [18] presented the Lightest Bin (LB) protocol that makes it harder for both non-colluding and colluding adversaries to manipulate the election of a leader to their advantage. While it already grants certain guarantees regarding the prevention of adversarial influence, it is also comparably simple. Therefore, we use a slightly modified version of the LB protocol for our theoretical approach and present

it in Section 4.4.1. It can be exchanged with more complex versions of the LB protocol that grant enhanced guarantees, for instance the protocol proposed by King et al. [30]. It improves the scalability of the LB protocol and guarantees the election of a good leader with a positive constant probability if less than one third of the candidates are adversarial. Furthermore, Kapron et al. [29] presented an asynchronous protocol that is based on the LB protocol and resilient to adversarial message scheduling.

2.11 Assumptions

We now introduce the assumptions on which we base our theoretical approach and the implementation. We refer to the underlying P2P network as the *underlay* as it interconnects all peers in the network. We formulate the following assumptions regarding the underlay:

- *Reachability*: The underlay is a connected graph.
- *No Churn*: No peer disconnects from the network.
- *Perfect Communication*: No packets are lost.
- *Authenticity*: All packets are authenticated.

Reachability ensures that the underlay is not partitioned, hence allowing communication between any pair of peers. No churn and perfect communication prevent the disruption of DC-nets by disconnecting peers and messages being lost. For a real-world application, perfect communication can be achieved by using reliable communication protocols, for instance, TCP for unicast and one of the techniques presented in Section 2.1 for multicast and broadcast communication. Authenticity prevents adversaries from impersonating other peers. As mentioned in Section 2.9, this also makes it impossible for adversaries to send conflicting information to different peers without it being detectable. Authenticity can be achieved in a real-world application by using unforgeable and verifiable signatures.

We introduce the usage of the following terms to prevent confusion:

- *peer*: a member of the underlay.
- *participant*: a member of a specified group (e.g., overlay, election and ADC-net).

We assume that each peer is *neighbor-aware* and hence knows all adjacent peers. This allows us to form groups by using distributed graph traversal. Furthermore, we assume that each peer can be *interested* in a number of topics. This is necessary for using interest-related overlays. While not actively sending or receiving messages, peers with empty interest sets can support anonymization techniques like ADC-nets by increasing the anonymity set size.

3 Related Work

In this chapter, we present past work that is related to our research. First, in Section 3.1, we refer to techniques that can be used for increasing the disruption-resilience of classical DC-nets. Then, in Section 3.2, we present protocols for anonymous communication that rely on DC-nets and address a number of challenges, including the ones presented in Section 2.6.4. Final, in Section 3.3, we refer to techniques for establishing overlays in a privacy-friendly manner.

3.1 DC-net Disruption Resilience

Waidner et al. [47] presented a protocol which achieves the goals of DC-nets under weaker assumptions. Furthermore, they showed that the DC-net disruption countermeasure proposed by Chaum [12] can be used by adversaries to evict non-adversarial participants from the communication. They presented improved protocols which prevent this attack.

Golle et al. [24] presented two protocols that can be used for detecting and identifying disrupting DC-net participants. Their protocols can be used for achieving full fault recovery from collisions induced by disrupting participants at a comparably low overhead of one additional broadcast round. Hereby, they increase the disruption-resilience of classical DC-nets.

3.2 DC-net Protocols

Goel et al. presented the Herbivore [23] protocol which addresses the DC-net overhead challenge. Instead of running one DC-net comprising all network peers, the network is partitioned into a number of cliques in which DC-nets are run. To prevent collisions, a reservation phase is used in which transmissions slots are assigned to the DC-net participants. They showed that their approach reduces the overhead induced by DC-nets and is also resilient to a number of critical attacks.

Corrigan-Gibbs et al. presented the Dissent [14] protocol which can be used for anonymous communication in groups with previously known members. It uses DC-nets for preplanned communication rounds in which collisions only occur if intentionally induced by adversarial participants. Such participants can be identified when they disrupt the communication and can then be evicted from groups.

3.3 Privacy-friendly Overlay Establishment

The pub-sub [17] scheme enables scalable group communication by decoupling senders and recipients of messages. Daubert et al. [15] formulated a number of privacy requirements for pub-sub systems and introduced a privacy-friendly overlay establishment technique. Their approach does not rely on a central trusted entity, hence allowing its application in P2P networks.

Grube et al. [25] introduced an approach for establishing pub-sub overlays using ant colony optimizations [16]. Their approach allows for a configurable trade-off between efficiency and privacy protection. They showed that this can be used for bridging the gap between privacy and efficiency.

4 Theoretical Approach

4.1 Introduction

The challenge being addressed in this thesis is the protection of sender-anonymity in a layered P2P network with many-to-many communication. We rely on the adversary model introduced in Section 2.3 and the assumptions formulated in Section 2.11.

This chapter is structured as follows: The traditional overlay approach and challenges with regards to sender protection are presented in Section 4.2. In Section 4.3, we show how DC-nets can be used in combination with overlays to address one of these challenges but at the same time introduces a number of new challenges. In Section 4.4, we present ADC-nets as modified version of DC-nets with reduced overhead. Final, in Section 4.5, we show how ADC-nets can be used in combination with overlays for solving the remaining challenges.

4.2 Traditional Overlay Approach

With the traditional overlay approach, a peer that wants to send a message is either already part of an overlay, joins an existing one or initiates a new one. As soon as it is part of the overlay, it can send messages to receivers on the overlay. This approach is highly problematic in terms of sender protection against our adversary model and leaves us with the following challenges:

1. *Sender Identification*: An adversary can identify the sender of the message by determining where the message was first forwarded.

2. *Overlay Interaction Observation*: An adversary can observe who joined or initiated an overlay directly before a message was sent over this overlay. This peer is with high probability the sender of the message.

Therefore, the traditional overlay approach grants no sender protection against our adversary model.

4.3 Overlays with DC-nets

The sender identification challenge introduced in Section 4.2 can be addressed by using DC-nets on top of overlays as shown in Figure 4.1. This way, overlay participants can use the DC-net for sending messages to receivers on this overlay. Hereby, eavesdropping adversaries cannot identify the sender of such messages. However, new challenges arise:

3. *DC-net Initialization Observation*: The DC-net has to be initiated by a peer. We assume that this peer initiates the DC-net because it wants to send a message. Hence, we encounter the same problems as for the overlay interaction observation challenge presented in Section 4.2.

4. *Overhead*: The enormous overhead of DC-nets has to be reduced before they can be practically used.

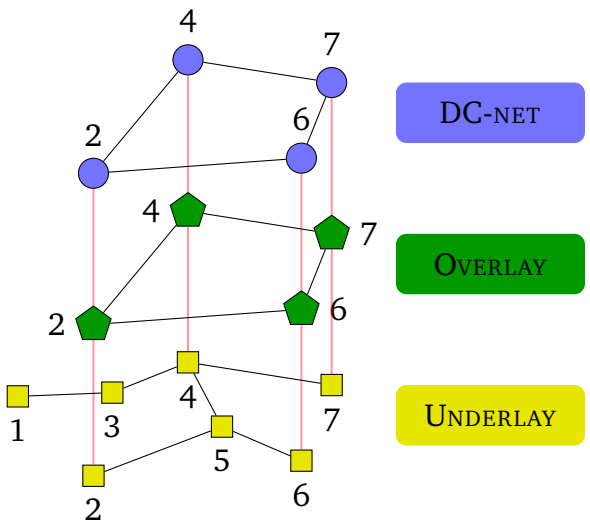


Figure 4.1.: Overlay with DC-net for sender protection.

5. *Overlay Churn*: DC-nets are not designed for dynamically joining and leaving participants. A DC-net has to be disbanded and newly created for every peer that joins or leaves the overlay on which the DC-net is running.

6. *Anonymity Set Size Limitation*: The anonymity set size of DC-nets is limited by the number of overlay participants. It may therefore be very small for less popular interests.

Furthermore, it should be noted that classical DC-nets rely on broadcast for message distribution. While this can also be used for this model, it may be a viable alternative to use multicast addressing in order to only address a subset of all peers, more specifically, the overlay members.

4.4 ADC-nets

We introduce a modified version of DC-nets that contain a rendezvous point (RP). A RP is a regular DC-net participant with additional tasks, namely

1. being the sole receiver of all DC-net messages of the DC-net,

2. decrypting the published secret messages and

3. forwarding them to their recipients that are not necessarily part of the DC-net.

We refer to DC-nets that contain a RP as ADC-nets due to the asymmetric message flow between the RP and the other participants. As published secret messages are sent to their recipients without protecting the anonymity of these recipients, receiver anonymity is not given for ADC-nets.

The structure of this section is as follows: First, in Section 4.4.1, we present how RPs can be elected. In Section 4.4.2, we present the communication model of ADC-nets and compare it to the one of DC-nets. While, this model reduces the overall communicational and computational overhead, we also show that certain problems are associated with it. Final, in Section 4.4.3, we discuss the possibilities of disbanding and pausing ADC-nets for further overhead reduction.

4.4.1 Rendezvous Point Election

In contrast to classical DC-nets, ADC-nets require the election of a RP before the actual communication can begin. Apart from this additional task, the initialization phase of ADC-nets is comparable to the one of DC-nets.

We formulate the following assumptions regarding the setting in which RP elections take place:

- Several peers have formed a group, for instance, by using the approach presented in Section 4.5.2, and initialize a new ADC-net for this group.
- Each of these peers knows every other peer in the group.

Prior to presenting the election scheme, we show the dangers of adversarial RPs. They can

- *eavesdrop* on published secret messages,
- *modify* messages before forwarding them and
- *drop* messages instead of forwarding them.

The problem of eavesdropping RPs is comparable to the problem of eavesdropping DC-net participants. It can be prevented by adding a layer of encryption to secret messages that can only be removed by recipients who know the decryption key.

Detecting the modification of messages is a new challenge that emerges when using ADC-nets instead of classical DC-nets. A RP cannot be prevented from modifying messages before forwarding them. However, it is possible to sign secret messages with a personal private key before adding an anti-eavesdropping encryption layer around it. Hereby, recipients can verify that a message came from a certain participant and was not modified in transit. However, adversarial recipients can use the signature to identify the sender of the message. Therefore, this technique is problematic with regards to sender anonymity and should only be used if all recipients are trusted.

Dropping of messages can be categorized as communication disruption. A RP can completely disrupt the communication by simply refraining from forwarding messages. We leave this challenge to future work as we do not focus on the aspect of disruption prevention.

The presented challenges makes the election of a honest ADC-net participant as RP desirable. Hence, the RP election should be robust and prevent both colluding and non-colluding adversarial participants from gaining an unfair advantage. At the same time, it is desirable to take certain aspects, e.g., the location of ADC-net participants in the underlay, into account. However, the inclusion of such aspects could be exploited by adversaries to boost the probability of them being elected as RP. Therefore, we use a randomized approach that is based on the LB [18] protocol.

A non-distributed version of our approach is presented by Algorithm 1. When executed in a distributed manner, each candidate $c \in C$ of the current round sends its vote to all other election participants, for instance, by using reliable multicast as presented in Section 2.1. Each election participant computes the candidate set for the next round after receiving all votes of the current round. If the new candidate set only contains one candidate, this candidate is elected as RP.

Using such a randomized approach is suitable with regards to sender anonymity as the outcome of an election is independent from the elected RP being a sender or not. Adversaries therefore cannot draw conclusions about a RP being a sender or not.

Algorithm 1 RP Election

$C \Leftarrow PARTICIPANTS$ ▷ set containing the current candidates
while $|C| > 1$ **do**
  $C_0 \Leftarrow \{\}$ ▷ 0-voting candidates
  $C_1 \Leftarrow \{\}$ ▷ 1-voting candidates
  for all $c \in C$ **do**
    if $random(0,1) == 0$ **then**
      $C_0.add(c)$
    else
      $C_1.add(c)$
    end if
  end for
  if $|C_0| > 0$ **and** $|C_1| > 0$ **then** ▷ candidates can only be eliminated if not unanimous
    if $|C_0| \leq \frac{|C|}{2}$ **then**
      $C \Leftarrow C_0$
    else
      $C \Leftarrow C_1$
    end if
  end if
end while
$\{RP\} \Leftarrow C$

Example 4.4.1: RP Election

Given a set $C = \{1,2,3,4,5,6,7,8\}$ of candidates that take part in a RP election.

Scenario 1: Fastest Election

Round	C	C_0	C_1	
1	$\{1,2,3,4,5,6,7,8\}$	$\{1,2,4,5,6,7,8\}$	$\{3\}$	$\Rightarrow$ Elected RP: 3

Scenario 2: Slowest Election (without unanimous rounds)

Round	C	C_0	C_1	
1	$\{1,2,3,4,5,6,7,8\}$	$\{1,6,7,8\}$	$\{2,3,4,5\}$	
2	$\{1,6,7,8\}$	$\{6,8\}$	$\{1,7\}$	
3	$\{6,8\}$	$\{6\}$	$\{8\}$	$\Rightarrow$ Elected RP: 6

Scenario 3: Election with Unanimous Rounds

Round	C	C_0	C_1	
1	$\{1,2,3,4,5,6,7,8\}$	$\{1,3,6,7,8\}$	$\{2,4,5\}$	
2	$\{2,4,5\}$	$\{\}$	$\{2,4,5\}$	
3	$\{2,4,5\}$	$\{2\}$	$\{4,5\}$	
4	$\{4,5\}$	$\{4,5\}$	$\{\}$	
5	$\{4,5\}$	$\{5\}$	$\{4\}$	$\Rightarrow$ Elected RP: 5

4.4.2 Communication

In contrast to DC-nets, ADC-net messages are not broadcasted to all peers in the network. Instead, every participant only sends its ADC-net messages to the RP of the ADC-net. The RP then decrypts the messages published in the ADC-net and forwards them to their actual recipients. This reduces both the number of messages sent and the number of XOR operations executed per round. This addresses the overhead challenge.

Assuming only one active sender per ADC-net, collisions, as presented in Section 2.6.4, do not occur and therefore do not pose a challenge for this communication model. However, when used with multiple senders, regular participants cannot detect anymore whether their messages were part of a collision as only the RP knows the published secret message of each round. This can be addressed by having the RP distribute the published secret messages not only to their actual recipients but also to all participants of the ADC-net. This results in additional communicational overhead that can be minimized by distributing the published messages via multicast. As shown in Section 6.2.2, even with collision detection, ADC-nets induce significantly less communication overhead than DC-nets.

4.4.3 Transiency

As shown in Section 6.2.2, the optimizations presented in Section 4.4.2 significantly reduce the overhead induced by DC-net CT. However, when compared to using no sender protection, each round still induces a considerable overhead. Especially when running over long periods of time and with only few non-empty secret messages being sent, the utilization of ADC-nets may be very low.

We propose the following techniques for reducing the overhead induced by ADC-nets that are not sufficiently utilized:

- *Pausing*: ADC-nets can be paused by temporarily stopping the computation and sending of ADC-net messages. Paused ADC-nets can be resumed, for instance, after a previously defined timespan.

- *Disbanding*: ADC-nets can be disbanded by permanently stopping the computation and sending of ADC-net messages. Furthermore, all information regarding the ADC-net, including cryptographic secrets, are deleted. Hence, disbanded ADC-nets cannot be resumed. We refer to ADC-nets that can be disbanded as *transient*.

While we discuss both techniques, we only implement and evaluate transient behavior and leave the practical evaluation of pausing to future work.

A number of challenges arise when using pausing and disbanding:

- *Low Utilization Detection*: A low ADC-net utilization has to be detected before pausing/disbanding can be initiated for the ADC-net.

- *Coordination*: ADC-net participants have to be informed of pausing/disbanding.

- *Transition Phase*: Pausing/Disbanding should only commence after a reasonable transition period in which the ADC-net participants can prepare for it.

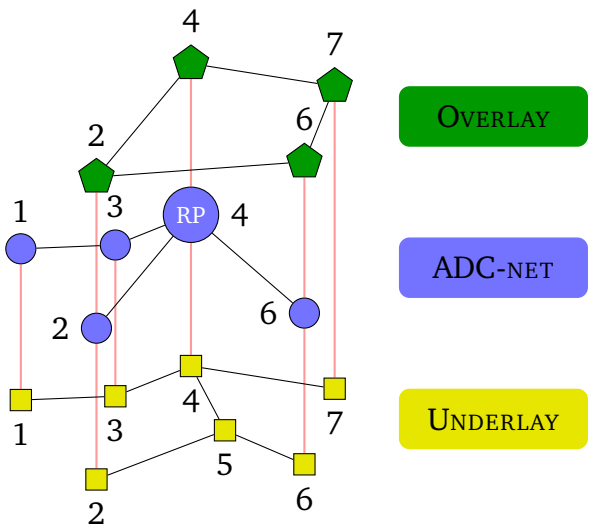


Figure 4.2.: Overlay with ADC-net for sender protection.

A variety of techniques for detecting a low ADC-net utilization are thinkable. We propose the following approach: The RP continuously measures the number of consecutive rounds in which only empty secrets messages were published. Hereby, it is predestined as coordinating entity and can initiate pausing/disbanding as soon as a certain threshold is exceeded. In Section 6.6, we analyze the impact of different threshold settings on the time until ADC-nets are disbanded.

The transition period before an ADC-net is paused/disbanded should be long enough to allow its participants to finish publishing their last messages. Messages that were not successfully published during the transition phase are most likely delayed. They can only be sent after the paused ADC-net is resumed or using another ADC-net if the original ADC-net was disbanded.

4.5 Overlays with ADC-nets

In this section, we introduce a new model that addresses the challenges introduced in the Sections 4.2 and 4.3. We use ADC-nets as intermediate layer between the underlay and multiple overlays instead of running DC-nets on specific overlays. An exemplary visualization of the model can be seen in Figure 4.2.

As shown in Section 6.2.2, ADC-nets induce less overhead than DC-nets. Hence, the overhead challenge is addressed by using ADC-nets instead of classical DC-nets.

We decouple ADC-net participants from overlay participants by using ADC-nets as intermediate layer between underlay and overlays. This decoupling addresses the overlay churn and the anonymity set size limitation challenges but at the same time introduces a new challenge:

7. *Group Formation*: Suitable ADC-net groups have to be formed.

The structure of this section is as follows: In Section 4.5.1, we discuss different ADC-net initialization techniques and address the DC-net initialization observation challenge which also applies to ADC-nets. Afterwards, we present our approach for setting up suitable groups in Section 4.5.2 and hereby address the group formation challenge. In Section 4.5.3, we address the overlay interaction observation challenge by using the RPs of ADC-nets as proxies for overlay interactions. Final, we present an example of the model in Section 4.5.4.

4.5.1 ADC-net Initialization

By definition, the DC-net initialization observation challenge introduced in Section 4.3 also applies to ADC-nets. Before addressing this challenge, we introduce the concept of *capacity* due to its relevance for our approach: We assume that each peer can take part in multiple ADC-nets at the same time. However, the participation in each ADC-net results in both communicational and computational overhead for a peer, especially if it is elected as RP. Therefore, peers can quickly be overloaded when the number of ADC-nets they can simultaneously take part in, is not limited. Hence, each peer is able to configure a capacity that limits the number of ADC-nets it can simultaneously participate in.

We address the DC-net/ADC-net initialization observation challenge by using a randomized approach for the initialization of ADC-nets. It can be configured using the following parameters:

- *Capacity*: the capacity of the peer.
- *Interval*: the interval in which the initialization check is executed.
- *Probability*: the probability for initializing a new ADC-net.

The approach is presented by Algorithm 2 and is executed individually on each peer. We now explain each phase:

- *Continuous Loop*: The peer periodically runs the initialization check.
- *Initialization Check*: The peer checks whether it has the capacities to initialize a new ADC-net. If it has and if the probability check succeeds, the peer initiates a new ADC-net.

Algorithm 2 Randomized ADC-net Initialization

```
▷ {Phase: Continuous Loop}
while True do
    → {Phase: Initialization Check}
    sleep(interval)
end while
```

```
▷ {Phase: Initialization Check}
if hasCapacities(peer) then
    if random() < probability then
        initiateADCNetGroupSetup()
    end if
end if
```

With this approach, the initialization of ADC-nets is randomized and independent from whether a peer wants to send a message. Hence, for each message sent through the ADC-net, the probability of it being sent by the ADC-net initiator is equal to it being sent by another ADC-net participant. This solves the DC-net/ADC-net initialization observation challenge.

4.5.2 Group Setup

The purpose of the group setup phase is the formation of a group of peers, for instance, by using one of the techniques presented in Section 2.7. It is required that each group participant knows every other group participant after the group setup phase. Only then can a RP be elected using the approach presented in Section 4.4.1.

Algorithm 3 Group Setup Technique

```
▷ {Configuration}
GSM ⇐ GROUP_SIZE_MINIMUM
GST ⇐ GROUP_SIZE_TARGET

▷ {Phase: Initialization}
peer ⇐ GROUP_SETUP_INITIATOR
path ⇐ [peer]
visited ⇐ [peer]
participants ⇐ [peer]
→ Phase: Forward Request

▷ {Phase: Forward Request}
cands ⇐ {n | n ∈ neigh(peer) ∧ n ∉ visited}
if |cands| > 0 then
    peer ⇐ randomChoice(cands)
    → Phase: Handle Request
else
    path.pop()
    if |path| > 0 then
        peer ⇐ path.pop()
        → Phase: Handle Request
    else
        if |participants| ≥ GSM then
            → Phase: Finish Setup
        else
            → Phase: Cancel Setup
        end if
    end if
end if

▷ {Phase: Handle Request}
path.append(peer)
if peer ∉ visited then
    visited.append(peer)
    if hasCapacities(peer) then
        participants.append(peer)
    end if
end if
if |participants| == GST then
    peer ⇐ path[0]
    → Phase: Finish Setup
else
    → Phase: Forward Request
end if

▷ {Phase: Finish Setup}
for all p ∈ participants do
    notifyFinish(p, participants)
end for

▷ {Phase: Cancel Setup}
for all p ∈ participants do
    notifyCancel(p)
end for
```

We now present a group setup technique which is comparably simple, fulfills the group setup requirements and therefore addresses the group formation challenge. To allow for a configurable anonymity set size, we introduce the following parameters that define a range of acceptable group sizes:

- *Group Size Minimum*: the minimum number of group participants required for a group setup to succeed.
- *Group Size Target*: the desired number of group participants.

The approach is presented by Algorithm 3. We now explain each phase:

- *Initialization*: The group setup is started by the peer acting as group setup initiator.
- *Forward Request*: the group setup request is forwarded to another peer in a random walk-like manner. An unvisited neighbor of the current peer is selected as next peer. However,

if no such neighbor exists, the group setup backtracks to the previous peer. If the group setup has already backtracked to the group setup initiator, it continues to the finish setup phase if the number of peers that have been recruited as participants equals or exceeds the group size minimum and otherwise to the cancel setup phase.

- *Handle Request*: The group setup arrives at a new peer. If this peer is visited for the first time, it checks whether to take part in the group and does so if it has enough capacities left. Then, the group setup either continues to the finish setup or the forward request phase, depending on whether the group size target has been reached.
- *Finish Setup*: The group setup initiator notifies all peers that agreed to participate in the group. Those peers are now participants of the new group and know each other participant of the group.
- *Cancel Setup*: The group setup initiator notifies all peers that agreed to participate in the group of its setup being canceled.

While not explicitly mentioned, each peer that decides to participate in a group setup reserves capacity for the case that it succeeds and results in a new ADC-nets. This reserved capacity is freed when the group setup is canceled. Hereby, it is ensured that the capacity is never exceeded.

Example 4.5.1: Group Setup

Assume a network with the peers $P = \{A, B, C, D\}$. Assume A to be directly connected to B, C and D. Assume B, C and D not to be directly connected to each other. Assume that each peer has enough capacities to take part in a new ADC-net.

Scenario: B initiates a group setup with a group size minimum of 3 and a group size target of 5. Assume that the handle request phase initiates a new round. Further assume round 0 as initialization phase. We present the state of the group setup after each round:

Round	*peer*	*visited*	*participants*	*path*	unvisited neighbors of *peer*
0	B	$[B]$	$[B]$	$[B]$	A
1	A	$[B,A]$	$[B,A]$	$[B,A]$	C, D
2	D	$[B,A,D]$	$[B,A,D]$	$[B,A,D]$	
3	A	$[B,A,D]$	$[B,A,D]$	$[B,A]$	C
4	C	$[B,A,D,C]$	$[B,A,D,C]$	$[B,A,C]$	
5	A	$[B,A,D,C]$	$[B,A,D,C]$	$[B,A]$	
6	B	$[B,A,D,C]$	$[B,A,D,C]$	$[B]$	

The group setup is finished because

1. all neighbors of B have already been visited and because
2. the group setup cannot backtrack further after reaching its initiator.

While the group size target of 5 has not been reached, the group setup still succeeds as the group size minimum of 3 is not undercut.

While our approach is easy to implement, it is problematic under the assumption of sabotaging insider adversaries. We encounter several problems that are similar to those presented in Section 2.6.3. Adversarial peers can

- *impersonate* other peers if they control all the communication links over which the random walk can reach these peers and
- *fake larger anonymity sets* by pretending to be connected to peers that do not actually exist.

The impersonation challenge can be addressed by using unforgeable and verifiable signatures for all communication messages. This way, each peer can check for every message whether it was actually sent by the apparent sender and discard it if it was not.

Preventing adversaries from faking larger anonymity sets is more problematic, especially in P2P networks where it can be assumed that each peer only knows a subset of the other peers. Hereby, honest peers cannot distinguish between real and faked peers. This problem is comparable to the sensor detection challenge in P2P botnets. Existing botnets address this challenge by limiting the number of botnet peers per subnet to prevent a high population of sensors [3]. While we do not evaluate this approach, it may be a suitable technique for preventing adversaries from faking a large number of peers.

While the adversary model introduced in Section 2.3 assumes that adversaries do not intentionally disrupt the communication, we still discuss this problem in the context of group setups: Adversaries can easily disrupt group setups by refusing to forward random walks to other peers. To increase the probability of at least one random walk not being disrupted by an adversary, we propose the execution of multiple random walks in parallel. This may also reduce the power of individual adversaries that try to fake participants to control a large portion of the resulting group. They have to "take over" multiple group setups that are executed in a random walk-like manner to prevent other peers from increasing the effective anonymity set size of the resulting group. However, the simultaneous execution of multiple random walks requires a modified version of Algorithm 3. We rely on a slightly modified version of the original algorithm with an additional timeout parameter. This parameter is used for stopping each random walk after a specific number of hops. Hereby, it can prevent

- the recruitment of too many participants and
- the traversal of all peers when only very few have capacities left.

Thus, each random walk can be executed independently from the other random walks without coordination by the group initiator. We evaluate the performance of using multiple random walks and timeouts on the basis of this approach in Section 6.3. It can certainly be refined, for instance, by using periodic coordination of the random walks to reduce the number of peers that are visited/recruited by multiple random walks. We leave such thinkable enhancements to future work.

4.5.3 Overlay Interaction

We now address the overlay interaction observation challenge introduced in Section 4.2. We propose that ADC-net participants conduct all sender-related overlay interactions through the ADC-net by using the RP as proxy. Hence, adversaries can only observe sender-related overlay interactions of RPs but cannot identify the ADC-net participants behind these interactions.

Sender-unrelated interactions, for instance, joining an overlay to receive messages on it, can still be executed directly by peers. This solves the overlay interaction observation challenge.

Our specific approach relies on *interest control messages* that contain one topic of interest and can be used by ADC-net participants to anonymously notify the RP of their interests. These messages are published in ADC-nets as secret messages. When a RP receives an interest control message, it tries to join an existing overlay that addresses the topic of interest. If no such overlay exists, the RP initiates a new overlay for this topic, e.g., by using one of the techniques presented in Section 3.3.

Using this approach, ADC-net participants can send their interests to the RP without being linked to them. This is crucial for sender protection as adversaries may use such information for identifying the sender of a message. Furthermore, interests should only be sent one-by-one to prevent adversarial RPs from linking ADC-net participants to their interests by comparing interest sets across ADC-nets.

Example 4.5.2: Dangers of Information Leakage

Given a set of peers $P = \{1, 2, 3, 4\}$. Assume that I_i represents the interest set of peer $i \in P$. Assume that $I_1 = I_2 = I_4 = \{\}$ and $I_3 = \{privacy, networking\}$. Assume that $A_{x,k} = \{j, k, l\}$ denotes the ADC-net with the identifier x, the RP $k \in P$ and the participants $j, k, l \in P$.

Scenario 1: Assume that the participants of $A_{a,1} = \{1, 2, 3, 4\}$ publish their interests. Assume that they send the interests in plain text to the RP instead of using the presented technique:

- 3 sends each interest in I_3 to the RP of $A_{a,1}$.
- A global eavesdropping adversary can identify the contents and the sender of these messages and hereby links I_3 to 3.
- 3 sends a privacy-related secret message over $A_{a,1}$ and the RP forwards it to its recipients on a privacy-related overlay.
- The global eavesdropping adversary can link this message to 3 as it is the only participant of $A_{a,1}$ that is interested in the privacy topic.

Scenario 2: Assume that 2 is an adversarial peer. Assume that the participants of $A_{b,2} = \{1, 2, 3\}$ and $A_{c,2} = \{2, 3, 4\}$ want to publish their interests not one-by-one but instead all interests at once:

1. 3 publishes I_3 in both $A_{b,2}$ and $A_{c,2}$.
2. As RP of both $A_{b,2}$ and $A_{c,2}$, 2 can detect that I_3 was published in both ADC-nets. As, apart from the adversary, 3 is the only peer that participates in both ADC-nets, 2 can link I_3 to 3 with high certainty.

4.5.4 Example

We present an example of our model in which we describe one of multiple scenarios that results in the network structure presented in Figure 4.2:

1. Peer 3 initializes the creation of the ADC-net after its initialization check, as presented in Section 4.5.1, succeeds.
2. Peer 3 starts the group setup as presented in Section 4.5.2. The outcome of this group setup is the set $G = \{1, 2, 3, 4, 6\}$ containing the group participants.
3. The RP election is conducted as presented in Section 4.4.1 and with G as initial candidate set. Peer 4 is elected as RP.
4. The ADC-net participants generate master secrets as presented in Section 2.6.1.
5. The ADC-net participants can now induce the creation of the overlay by sending their interests to the RP as presented in Section 4.5.3. For simplicity, we assume that all peers in G are interested in the same topic and that the created overlay is related to this topic.
6. We assume that the remaining participants of the overlay joined it after its initialization by the RP.
7. Now, each participant of the ADC-net can use it to send messages to every participant of the overlay by using the RP as proxy as presented in Section 4.4.2.

5 Implementation

5.1 Introduction

We developed a prototypical framework for demonstrating and evaluating our theoretical approach. We chose Python as programming language for the implementation and elaborate on this selection in Section 5.2.

Our framework comprises two main packages and several scripts:

- The `adc_net` package contains the main logic for running simulations and hence can be seen as the main application. The different components of this package are presented in Section 5.3.

- The main application is supplemented by several scripts that can be used for different purposes, including the deployment of the framework, the execution of simulations with configuration presets and the interaction with running simulations. The different scripts are presented in Section 5.4.

- The `adc_net_eval` package contains the logic for evaluating simulation runs and is presented in Section 5.5.

Throughout this chapter, we present a number of modules and scripts that can be configured via command-line interfaces (CLIs). Detailed information regarding the specific arguments and flags can be accessed by calling the respective script with the `--help` flag. Therefore, we mainly present the functionality and only sometimes refer to the specific arguments and flags. Furthermore, we only present the most relevant options.

5.2 Python

Python is an interpreted high-level programming language that is especially popular in the scientific domain. We present a number of reasons for Pythons popularity and also discuss the relevance of certain aspects for our implementation in Section 5.2.1. As can be expected, Python also has certain limitations when compared to other programming languages. In Section 5.2.2, we discuss one specific problem that has an impact on our implementation. Last, we list the Python community packages our framework depends on in Section 5.2.3.

Design 5.2.1: Python Version

As of writing, Python 3.5 is the latest version for which official Raspbian Stretch Python packages are available. Therefore, we use this Python version to allow the execution of distributed simulations using multiple Raspberry Pis.

5.2.1 Positive Aspects

When compared to other high-level programming languages like C or Java, the simplicity and readability of Pythons syntax stands out. Still, it is a very powerful language with a reasonable abstraction level. Due to its popularity, it is widely supported by different operating systems. Combined with Python being an interpreted language, this allows for the easy deployment of our framework. Furthermore, Pythons large standard library allowed us to develop a complex framework within a reasonable timespan. It is supplemented by a multitude of community-maintained packages that can easily be installed in a Python virtual environment. We depend on several community packages and list them in Section 5.2.3.

5.2.2 Negative Aspects

CPython is the reference implementation of the Python interpreter and hence widely used. It uses a global interpreter lock (GIL) [2] which is a mutex that ensures that only one thread within a process can execute code at a time. While this ensures the thread-safety of certain Python operations, it prevents the effective use of multithreading. With our framework being designed for running local simulations with more than 1,000 peers of which each one runs multiple threads, the GIL becomes a critical problem. It is unfeasible to run simulations with this number of peers on only one CPU core.

Due to multithreading not being a viable alternative, we resort to multiprocessing where each peer has its own process. While running multiple Python threads is relatively lightweight due to many resources being shared among different threads, Python processes cause a higher memory usage due to the strict isolation of resources. We observed the impact of using multiprocessing instead of multithreading on the memory usage of our simulations and present our results in Table 5.1. The different peers were only initialized and no actual simulation logic was executed. We conducted three tests per setting that yielded very similar results and used the median value for the computation of the approximate memory usage per peer. It can be seen that the memory usage increases significantly when using multiprocessing. However, we put up with the increased resource requirements as it allows us to fully utilize modern multi-core systems and conduct real-time simulations with a considerable number of peers.

Table 5.1.: Comparison of the memory usage of initialized `adc_net` simulations for multithreading and multiprocessing. All of these simulations were conducted on the same machine running Arch Linux (AMD64).

Peers	Total Memory Usage		Approximate Memory Usage per Peer	
	M.Threading	M.Processing	M.Threading	M.Processing
100	0.05G	0.61G	0.50M	6.10M
500	0.13G	3.14G	0.26M	6.28M
1000	0.25G	6.38G	0.25M	6.38M
2000	0.48G	13.43G	0.24M	6.72M

5.2.3 Community Packages

Our implementation depends on the following community packages:

- `cryptography`: cryptographic library package. Used for key exchanges and key derivation in the `adc_net/util/crypto` wrapper module presented in Section 5.3.4.
- `networkx`: network generation and interaction package. Used for the generation of network definitions in the `network_generation` script presented in Section 5.4.2.
- `paramiko`: SSH connection package. Used for remote deployment in the `remote_deploy` script presented in Section 5.4.1.
- `numpy`: array-processing package. Used for the evaluation in the `adc_net_eval` package presented in Section 5.5.
- `scipy`: scientific library package. Used for the evaluation in the `adc_net_eval` package presented in Section 5.5.
- `plotly`: plotting package. Used for the generation of evaluation plots in the `adc_net_eval` package presented in Section 5.5.

5.3 Main Application: `adc_net` Package

The `adc_net` package contains the functionality for running simulations and hence can be seen as our main application. We designed it with two main modes:

- `simulation`: All peers are simulated on one host using loopback addresses.
- `prototype`: Peers are simulated on multiple hosts in a local network.

Both modes rely on the same code base and only differ in the way that simulation peers are set up. To prevent confusion, we refer to simulations run in the `simulation` mode as *local* and to simulations run in the `prototype` mode as *distributed*.

The structure of this section is as follows: First, in Section 5.3.1, we present the `__main__` module which functions as entry module for the `adc_net` package. In Section 5.3.2, we present the `conf` module which can be used for the configuration of simulations. In Section 5.3.3, we present the `comm` module which is used for low-level communication between peers. In Section 5.3.4, we present the `crypto` and the `logging` modules which function as wrapper modules for the `cryptography` community package and the official Python `logging` package. In Section 5.3.5, we present the `messaging` module which is used for building and parsing standardized program messages. In Section 5.3.6, we present the `peer` module that combines the functionality of the other modules in the `Peer` class which is used for simulating a peer. In Section 5.3.7, we present the `handler` package which is used for executing group setups, RP elections and for running ADC-nets. Final, in Section 5.3.8, we present the `evaluation` package which is used for logging evaluation information that can then be analyzed by the `adc_net_eval` package presented in Section 5.5.

5.3.1 Program Entry: `__main__.py`

The `__main__` module can be used for starting `adc_net` simulations. As such, it is responsible for

- loading the configuration from the file system using the `conf` module presented in Section 5.3.2,
- setting up debug logging using the `logging` module presented in Section 5.3.4,
- setting up evaluation logging using the evaluation package presented in Section 5.3.8 and
- initializing one or multiple simulation peer(s) depending on the used main mode. Peers are initialized by instantiating the `Peer` class presented in Section 5.3.6.

5.3.2 Configuration: `conf.py`

The `conf` module is used for reading configurations from the file system and for parsing them. We use the JSON format for configurations as it can easily be imported as python dictionary.

We use the following types of configurations:

- *Main*: allows the configuration of most program settings.
- *Prototype*: allows the configuration of a single simulated peer.
- *Simulation*: allows the configuration of multiple simulated peers.

All the settings are listed and explained in the `README` of the project. Therefore, we only list the most relevant settings of the main configuration in the Tables A.1, A.2, and A.3. The prototype and simulation configurations mainly differ in the number of peers being configured. While the prototype configuration only contains one peer, the simulation configuration contains a list of peers. The peer configuration options are the same for both configuration types and listed in Table A.4.

For each type of configuration, a default configuration exists. They contain settings that proved to be reasonable defaults. Each default configuration can be overwritten by using a local configurations.

Configurations can also be supplied as function parameters to the `main` function of the `__main__` module presented in Section 5.3.1. This allows an easy integration with the `batch_execution` script presented in Section 5.4.4 as modified configurations can directly be used for executing `adc_net` simulations without the need for intermediate file system interactions.

5.3.3 Network Communication: `comm.py`

The `comm` module is used for low-level network communication between peers. We use UDP as communication protocol as it is connectionless and hence easy to use in a P2P setting.

We use two designated threads:

1. *Sender Thread*: Continuously fetches messages from the *send queue*, serializes and sends them from the address of the current peer.

2. *Receiver Thread*: Listens on the address of the current peer on the configured port, deserializes received messages and puts them into the *receive queue*.

This approach allows us to execute blocking network operations without blocking the peer logic.

While we keep our implementation simple, we propose the following thinkable improvements for real-world applications:

- *Multicast*: We only implement unicast communication. Multicast could be used for improving the performance of group communication operations. For instance, the distribution of RP election votes could be realized more efficiently using multicast.
- *Signatures*: As mentioned in Section 2.11, unforgeable and verifiable signatures could be used for preventing adversaries from impersonating other peers. This requires a public key infrastructure (PKI) which can be entrusted with the distribution of the public keys of each peer to allow the verification of signatures. Assuming the existence of such a PKI, the signing and verification of messages could be implemented on a low level in the `comm` module as follows:
  - Each outgoing messages is signed with the private key of the sending peer.
  - The signature of each incoming message is verified using the public keys distributed by the PKI. If the verification fails, the message is discarded.

5.3.4 Utilities: `util` Package

The `util` package contains wrapper modules that encapsulate functionality from the Python standard library and community packages.

Cryptography Wrapper: `crypto.py`

The `crypto` module provides functions for cryptographic operations used throughout the `adc_net` module. As cryptography is complex, we rely on the `cryptography` community package for the implementations of low-level cryptographic operations.

Design 5.3.1: Elliptic-curve Cryptography

Elliptic-curve cryptography (ECC) can be used for achieving a comparable level of security with significantly smaller key sizes than classical asymmetric cryptography schemes like RSA [33]. Therefore, we use ECC for cryptographic operations.

As mentioned in Section 5.2, we wanted to allow the execution of distributed simulations using multiple Raspberry Pis. To allow the efficient execution of cryptographic operations on Raspberry Pis (Model 3B), we conducted a number of tests using the OpenSSL speed command [1] and selected SECP256R1 as elliptic curve. Our tests indicate that this curve is a good trade-off between security and performance for our specific use case.

The `crypto` module provides the following cryptographic functionality:

- the generation of asymmetric key pairs using ECC.

- the generation of shared master secrets for the ADC-net communication using pair-wise elliptic-curve Diffie-Hellman (ECDH) key exchanges. Both peers use their private key and the public key of the other peer for the ECDH key exchange.
- the derivation of round secrets using HKDF [31] as key derivation function (KDF). Round secrets for a specific master secret and a specific round are derived by using the master secret as input key material and the round number as salt.

While we keep our implementation simple, we propose the following thinkable improvements for real-world applications:

- Authenticated ECDH should be used to prevent man-in-the-middle (MITM) attacks. This can be achieved by using verifiable and unforgeable signatures for all communication messages as proposed in Section 5.3.3.
- As proposed in Section 2.6.3, new master secrets could be periodically generated to improve the security of the ADC-net communication.

Logging Wrapper: `logging.py`

The `logging` module functions as wrapper for the Python logging module and is used for debug logging. Whether debug logging is enabled and whether log messages are written to a log file or the standard output `stdout` can be configured in the `adc_net` main configuration. Since debug logging is not relevant for the actual functionality, we do not explain this module in detail.

5.3.5 Messaging: `messaging` Package

The `messaging` package is used for building and parsing standardized *program messages* that are used for the communication between different peers. We use python dictionaries for all program messages as they allow us to use multiple key-value entries.

Each program message contains a *type* key whose value is an element of the *MsgType* enumeration which contains all program message types. Therefore, each peer that receives a message can determine the type of this message and handle it accordingly as presented in Section 5.3.6. The other entries of program message dictionaries vary for the different message types. For instance, each group setup program message contains the `group_uid` entry which is used for distinguishing between different group setups, RP elections and ADC-nets as presented in Section 5.3.7.

5.3.6 Peer Logic: `peer.py`

The `peer` module contains the `Peer` class. This class is used for simulating a peer and hereby combines the logic from the other `adc_net` modules. As part of the initialization of a `Peer` instance, the following tasks are executed:

1. The interest set and the neighbor list of the peer are stored.
2. The communication component of the peer is initialized. The functionality of this component is presented in Section 5.3.3.

3. The approach logic handlers presented in Section 5.3.7 are initialized. They are used for the logic behind group setups, RP elections and ADC-net functionality.

4. The following threads are initialized:
   - *Message Handler Thread*: processes received program messages.
   - *Periodic Handler Thread*: executes certain tasks periodically.

   We use these two different threads as their tasks contain blocking operations, hence rendering a sequential implementation impractical. Both threads continuously execute their tasks until the peer receives a *shutdown* program message and shuts down. Shutdown messages are explained in the next section.

Message Handler Logic

The message handler continuously fetches received program messages from the receive queue presented in Section 5.3.3 and processes each of them as follows:

1. The type of the program message is parsed using the `messaging` package presented in Section 5.3.5.
2. The other entries of the program message are parsed in dependency to its type using the `messaging` package.
3. The handler function designated for this program message type is called with the parsed information.

Most of the handler functions are part of one of the `handler` modules presented in Section 5.3.7. However, two special types of program messages are directly handled in the message handler of the peer:

- *Startup*: When a peer first receives a startup program message, it forwards this message to all its neighbors and then starts with the periodic execution of tasks. Subsequent startup program messages are ignored.
- *Shutdown*: When a peer receives a shutdown program message, it forwards this message to all its neighbors and then shuts down by terminating its handler threads.

Design 5.3.2: Startup & Shutdown Program Messages

We use program messages for starting and stopping simulations. Using the presented approach, we can easily start/stop both local and distributed simulations by sending a single startup/shutdown program message to one of the simulation peers. For this purpose, we created the `startup` and `shutdown` scripts that are presented in Section 5.4.5.

Before using this approach, we encountered the *Startup Phase Message Loss* challenge. We observed that it can lead to message loss if peers start group setups and forward them to neighbor peers that are not yet set up. Our approach solves this challenge as we can observe the initialization status of the simulation peers and start the simulation as soon as all peers are set up. As shown in Section 5.4.4, this can also partly be automatized.

Periodic Handler Logic

The periodic handler periodically executes tasks. We essentially have two types of tasks:

- The (ADC-net) group setup initialization check presented in Section 5.3.7.
- Periodic ADC-net tasks that are executed for all ADC-nets in which the peer participates, for instance, the periodic sending of ADC-net messages.

As presented in the last section, the periodic handler only starts with the periodic execution of tasks after its peer has received a startup message. This addresses the startup phase message loss challenge.

5.3.7 Approach Logic: `handler` Package

The main logic of our theoretical approach, which is presented in Chapter 4, is implemented in the `handler` package. We implemented group setups, RP elections and ADC-net functionality in different modules and present them in the following sections.

Group Setup: `group.py`

The `group` handler module contains the logic for executing distributed group setups as presented by Algorithm 3. The communication for these setups is realized using a number of group setup program messages. We do not explain the different group setup program messages and their handling in detail as the logic is very similar to the group setup approach presented in Section 4.5.2.

It is desirable that multiple group setups can be executed in parallel in the same network without this potentially leading to identification problems. This requires a unique group identifier that can be used for linking each group setup program message to its respective group setup.

Design 5.3.3: `group_uids`

We use a tuple containing

1. the address of the peer that initiated the group and
2. a timestamp of the time at which the group was initiated

as (sufficiently) unique identifier of a group and refer to it as `group_uid`. We not only use it for distinguishing between multiple group setups but also for uniquely identifying RP elections and ADC-nets as they are all executed in a specific group. This allows the parallel execution of multiple group setups, RP elections and ADC-nets in the same network.

We decided to use Unix epoch timestamps as they can be easily queried using the Python `time` package. However, we observed problems with using the standard time resolution of seconds since the Unix epoch. In very few cases, multiple groups were initiated by the same peer within a very short timespan, resulting in the same `group_uid` being used

for these groups. Therefore, we use the number of milliseconds since the Unix epoch as timestamps to reduce the probability of duplicate `group_uids`. We did not observe any duplicate `group_uids` for the increased time resolution.

The module also includes the logic for periodically checking whether a new ADC-net should be initialized as such an initialization starts a new group setup. This check is implemented like the initialization check presented by Algorithm 2. It is periodically executed by the periodic handler of the peer as presented in Section 5.3.6.

RP Election: `election.py`

The `election` handler module contains the logic for executing distributed RP elections comparable to the non-distributed version presented by Algorithm 1. We refer to Section 4.4.1 for a detailed description of the used approach.

ADC-net Functionality: `adc.py`

The `adc` handler module contains the logic for

1. finalizing the initialization of ADC-nets by generating master secrets and
2. ADC-net runtime functionality.

For each ADC-net, master secrets are generated using ECC as presented in Section 5.3.4. We refrain from a detailed description of the master secret generation as it can be exchanged with any other technique that establishes master secrets as presented in 2.6.1.

We refrain from describing each implementation aspect of the ADC-net functionality that is presented in detail in Section 4.4. Instead, we present and explain a number of design decision we made during the implementation:

- The theoretical approach presented in Section 4.4 does not specify when the decryption of ADC-net messages is conducted by RPs.

  Design 5.3.4: ADC-net Message Decryption

  In our initial implementation of the ADC-net message decryption, a RP waited until it had received all ADC-net messages of a round and then decrypted the published secret message by XORing all ADC-net messages of this round. Especially for larger groups, this is problematic as a lot of ADC-net messages have to be stored by the RP before the published message can be extracted by XORing all the ADC-net messages.

  We address this challenge by using *on-the-fly decryption*. Here, the RP directly XORs the ADC-net messages of a round upon their arrival and only stores the result of the consecutive XOR operations instead of each ADC-net message. Furthermore, for each round, the RP keeps track of the ADC-net participants from which an ADC-net message has been received. Thereby, it can detect whether all messages have been received and also prevent disruption by participants that send multiple ADC-net

messages for the same round.

On-the-fly decryption reduces the memory requirements for each RP: Only one message and tracking information have to be stored for each round for which not all ADC-net messages have yet been received. Keeping track of the ADC-net participants whose ADC-net message has been received can be done efficiently, for instance using an array structure in which one bit is used for representing the state of each participant.

- In Section 2.6.4, we presented the challenge of colliding non-empty secret messages. In Section 4.4.2, we presented an optional modification of the ADC-net communication model for allowing collision detection. Hence, approaches for collision detection and handling that can be applied to DC-nets can also be applied to ADC-nets.

Design 5.3.5: Collision Handling

We implemented collision handling as optional functionality which can be en-/disabled using the `adc.collision_handling.enabled` configuration option listed in Table A.3. We use the following technique which is comparable to slotted ALOHA [42]:

  - If a peer detects that a non-empty secret message that it previously tried to publish in an ADC-net was part of a collision, it initiates a *wait period* within this ADC-net.
  - During this wait period, the peer only sends empty secret messages in the ADC-net to prevent further collisions.
  - The duration of the wait period is randomized and depends on the `adc.collision_handling.wait_prob` configuration option listed in Table A.3. During the wait period, a probability check is conducted each ADC-net round to determine whether the wait period continues.
  - As soon as the wait period ends, the peer retransmits the non-empty secret message whose collision resulted in the wait period.

- We wanted to be able to simulate the sender behavior of peers to allow more realistic simulations and a more detailed evaluation.

Design 5.3.6: Sender Control and Message Tracking

We decided to use program messages for controlling the send behavior of peers. For this purpose, we created the `sender_control` script which is presented in Section 5.4.5.

When a peer receives a sender control message, it adopts the *send configuration* defined by this message for a random ADC-net it takes part in. We do not explain

the periodic generation of messages for sender configurations in detail and instead refer to the `sender_control` script configuration options listed in Table A.6.

In addition to regular message sending, we also implemented *message tracking*. If message tracking is enabled, the following information is added to each message that is generated by sender configurations:

- *Sender*: the sender of the message.
- *Round*: the number of the ADC-net round in which the message was generated and added to the send queue.

This option is clearly leaking sensitive information regarding the sender and as such should never be used for a real-world applications. However, it allows us to evaluate certain aspects, for instance, the delay between a message being generated and it being successfully published in the respective ADC-net. We use this option for tracking the mean delay of messages for the evaluation presented in Section 6.5.

- In Section 4.4.3, we presented transient behavior as option for reducing the overhead induced by ADC-nets by disbanding ADC-nets that are not sufficiently utilized.

Design 5.3.7: Transiency

We implemented transient behavior as optional functionality which can be en-/disabled using the the `adc.transiency.enabled` configuration option listed in Table A.3. This table also contains a number of options for configuring the different aspects of transiency which are presented in Section 4.4.3.

- In Section 4.5, we presented how ADC-net participants can use the RP as proxy for overlay interactions.

Design 5.3.8: Overlay Interactions

For simplicity, we did not implement the overlay interactions executed by RPs. Instead, RPs log the secret messages that are published in their ADC-net. These secret messages also include interest control messages that are presented in Section 4.5.3.

5.3.8 Evaluation: `evaluation` Package

The `evaluation` package contains functionality for logging information that is relevant for evaluation purposes. This is conducted as follows:

- We use *evaluation messages* for logging. Their format is comparable to the one used for program messages.
- Each peer logs its evaluation messages to a CSV log file designated for this peer. Hereby, the same evaluation logging logic can be used for local and distributed simulations.

- A peer logs an evaluation message when an evaluation-relevant event occurs. We do not discuss these events as this goes beyond the scope of a short summary of the `evaluation` package.
- Each peer also creates a JSON metadata file containing the configuration options used for the simulation.

As presented in Section 5.5, this information can be used for reconstructing and evaluating group setups, RP elections and ADC-nets.

5.4 Scripts

We created a number of scripts for different tasks and present them in this section.

5.4.1 Deployment: `deploy.py`, `remote_deploy.py`

We wrote two scripts to allow the easy deployment of our framework:

- The `deploy` script can be used for
  - cloning the repository from a central Git repository,
  - pulling the latest version of the program code from this repository,
  - setting up a local Python virtual environment and
  - installing the required community packages in this virtual environment.
- The `remote_deploy` script can be used for
  - copying files required for the deployment to a remote host via SFTP and
  - executing the `deploy` script on a remote host via SSH.

The `deploy` script relies on a private SSH key which only allows read access to the central Git repository and is stored in the repository itself. This way, the `deploy` script can execute Git operations using SSH without the need for additional SSH configuration on the host. While it is never a good idea to publish private keys in repositories, it is acceptable for this use case as only read access is granted.

The `remote_deploy` script uses SSH for connecting to remote hosts. It was primarily designed for the deployment to Raspberry Pis located in a local network and therefore uses the Raspberry Pi default credentials for the SSH login.

5.4.2 Network Generation: `network_generation.py`

The `network_generation` script can be used for generating network definitions with configurable properties. It can be configured via a CLI using the options presented in Table A.5. As the options are explained in this table, we refrain from presenting them here. However, to prevent confusion, we introduce the term *node* which we use when referring to peers whose network properties are defined but that are not instantiated in a simulation.

5.4.3 Group Setup Simulation: `group_setup_simulation.py`

The `group_setup_simulation` script can be used for simulating group setups with extended functionality without having to integrate the extensions into the group setup logic of our `adc_net` application. We created this script for conducting the evaluation presented in Section 6.3. For simplicity, we implemented a non-distributed version of the group setup algorithm presented in Section 4.5.2 which can be configured using the following group setup configuration options:

- *Number of Walks*: the number of random walks that are executed in parallel.
- *Group Size Target*: the desired number of group participants.
- *Participation Probability*: the probability of a peer being recruited as participant when it is first visited by a random walk. This option functions as simple replacement of the capacity configuration and can be used for simulating the percentage of peers in a network that have reached their capacity.
- *Timeout*: the number of hops at which a random walk is finished regardless of the number of recruited participants.

A CLI can be used for supplying the group setups configuration options as arguments. Additionally, the following aspects can be configured via the CLI:

- the properties of the network used for the group setup simulations. The network is generated using the network generation script presented in Section 5.4.2.
- the number of simulations that are executed for the specified options.
- whether the exhaustive simulation mode is enabled. It is used for the group setup evaluation presented in Section 6.3.

We designed the evaluation mode for executing a considerable number of simulations for each combination of the group setup configuration options. Thereby, we use multiprocessing for the parallel execution of multiple simulations to fully utilize multi-core systems and reduce execution time.

5.4.4 Controlled ADC-net Execution: `batch_execution`, `batch_evaluation`

The `batch_execution` script can be used for executing `adc_net` simulations for a number of hard-coded configuration presets. It therefore functions as wrapper for the `adc_net/__main__` module presented in Section 5.3.1. A number of configuration presets for the following configuration types exist:

- *RunMode*: defines whether local or distributed simulations are executed.
- *MainConfMode*: defines a number of modifications to the default `adc_net` main configuration.
- *NetworkConfMode*: defines the properties of the network used for the simulations. The definition of the simulation network is generated using the `network_generation` script presented in Section 5.4.2.

- *ControlMode*: defines interactions with a running simulation.

In addition to hard-coded configuration presets,

- the *number of simulations* executed consecutively and
- the *duration of each simulation* in seconds

can be set. The modes and configuration options can be configured via a CLI.

Design 5.4.1: Sender Control Modes

Control modes can be used for interacting with running simulations. While they can be used for all kinds of interactions, we only use them for controlling the send behavior of peers: We use *Sender Control Modes* defined by a tuple of the form (*sender rate*, *send probability*) with

1. the *sender rate* as percentage of peers that are potential senders and
2. the *send probability* as probability with which a potential sender sends a non-empty secret message in an ADC-net round.

Sender control modes are realized by sending sender control program messages to a number of randomly selected peers after starting the simulation via a startup message. The number of peers depends on the sender rate and the configuration of the sender control program messages depends on the send probability.

We use sender control modes for the evaluation of our ADC-net collision handling and ADC-net transiency implementations in the Sections 6.5 and 6.6.

Design 5.4.2: Simulation Startup and Duration

As presented in Section 5.3.7, simulations should only be started when all simulation peers are set up. Therefore, we use the following techniques for ensuring that all peers are set up:

- *Local Simulations:* We use multiprocessing barriers that are passed when all simulation peers are set up.
- *Distributed Simulations:* We use a startup wait time parameter that has to be set manually and defines the time after which each distributed peer has to be set up.

When all peers are set up, the simulation is started using startup program messages as presented in Section 5.3.6. Also, a threaded timer is initialized which initiates the sending of shutdown program messages after the specified simulation duration.

The `batch_evaluation` script functions as wrapper for executing `adc_net` simulations for the evaluation for a number of configuration combinations. Its CLI can be used for selecting

- which type of evaluation simulation is run and
- how many simulations are run for each configuration combination.

We used this script for running the simulations for the evaluation tasks presented in the Sections 6.4, 6.5 and 6.6.

5.4.5 Simulation Interaction

We created a number of scripts for interacting with running simulations and present them in this section.

Simulation Startup: `startup.py`

The `startup` script can be used for sending a startup program message to a peer. This peer can be specified via a CLI by supplying an address and a port. Startup messages are presented in Section 5.3.6.

Sender Control: `sender_control.py`

The `sender_control` script can be used for sending sender control program messages to a peer to control its sender behavior. The different options that can be supplied to the script via a CLI are listed in Table A.6. When the script is executed, a sender control program message containing the specified configuration settings, except the addressing information, is sent to the addressed peer. How sender control program messages are handled by peers is described in Section 5.3.7.

Simulation Shutdown: `shutdown.py`

The `shutdown` script can be used for sending a shutdown program message to a peer. This peer can be specified via a CLI by supplying an address and a port. Shutdown messages are presented in Section 5.3.6.

5.5 Evaluation Application: `adc_net_eval` Package

The `adc_net_eval` package can be used for evaluating group setups, RP elections and ADC-nets simulated with the `adc_net` package presented in Section 5.3. We only present the general evaluation procedure and refrain from explaining the specific tasks and the CLI options as this goes beyond the scope of a short summary of the `adc_net_eval` package:

1. The `adc_net` evaluation logs for either a specified or all simulation runs are read in from the file system, parsed and used for reconstructing the group setups, RP elections and ADC-nets on-the-fly.
2. The reconstruction of the group setups, RP elections and ADC-nets is finalized.
3. The reconstructed information is used for evaluating a number of aspects.
4. *(optional)* Evaluation plots are created and exported to the file system.
5. *(optional)* The evaluation results are exported as JSON to the file system.

We use multiprocessing for the parallel evaluation of multiple simulation runs to fully utilize multi-core systems and reduce execution time.

6 Evaluation

6.1 Introduction

In this chapter, we evaluate our theoretical approach and our implementation of this approach. Furthermore, we discuss the results of this evaluation. The structure of this chapter is as follows: First, we compare DC-nets to ADC-nets with regards to their anonymity set size and overhead in Section 6.2. Then in Section 6.3, we analyze the impact of different group setup configuration parameters on a number of group setup properties using the `group_setup_simulation` script presented in Section 5.4.3. In Section 6.4, we evaluate the speed of our RP elections and the distribution of elected RPs in the underlay using our `adc_net` implementation. We also use it for analyzing the performance of our ADC-net collision handling technique in Section 6.5 and for evaluating the impact of different transition threshold settings on transient ADC-nets in Section 6.6. We conclude the evaluation by addressing a number of relevant research questions in Section 6.7.

For the evaluations in the Sections 6.3 to 6.6, we ran a number of simulations with different goals and experimental setups. Therefore, we structure these sections as follows:

1. *Approach:* We present our approach and introduce the goals of the evaluation.
2. *Experimental Setup:* We describe the experimental setup used for the simulations.
3. *Results:* We present and discuss the evaluation results.

6.2 Comparison of DC-nets and ADC-nets

We presented classical DC-nets in Section 2.6 and introduced ADC-nets in Section 4.4. In Section 6.2.1, we compare them with regards to their anonymity set size. Then, in Section 6.2.2, we compare their communication and computation overhead.

6.2.1 Anonymity Set Size

We introduced anonymity sets in Section 2.2 and mentioned that their size can be used as anonymity metric. We also presented how colluding adversaries may reduce the effective anonymity set size of DC-nets/ADC-nets. Therefore, larger anonymity sets do not necessarily exhibit a better privacy protection than smaller anonymity sets. However, we assume that

Table 6.1.: The anonymity set size for send operations for different communication models in a network with P as set of peers and P_{ADC} as set of ADC-net participants.

Communication Model	Anonymity Set Size
DC-nets	$\|P\|$
ADC-nets	$\|P_{ADC}\|, P_{ADC} \subseteq P$

Table 6.2.: Communication (m) and computation (o) overhead per round of communication for different communication models. Assume p as number of DC-net/ADC-net participants and r as mean number of recipients per round to which a RP has to forward a published message.

Communication Model	m	o
DC-nets	$p \cdot (p-1)$	$p \cdot (p-1) + p \cdot (p-1)$
ADC-nets	$(p-1) + r$	$p \cdot (p-1) + p - 1$
ADC-nets with collision detection	$2 \cdot (p-1) + r$	

larger anonymity sets always exhibit a better privacy protection to simplify the following comparison, which is visualized in Table 6.1:

- *Classical DC-nets* comprise all peers. Hence, they always exhibit the maximum anonymity set size that can be achieved in the underlying network.
- *ADC-nets* comprise a configurable number of peers when used as presented in Section 4.5. Hence, the anonymity set size can be individually selected for specific use cases. For instance, communication with critical content can be protected by using larger anonymity sets at the cost of increased overhead.

Yet, it has to be noted that the anonymity set size of ADC-nets only applies for send operations. Due to our overhead reduction optimizations, receiver anonymity is not preserved anymore. However, this is not problematic for our approach as we only focus on sender protection.

6.2.2 Overhead

As mentioned in Section 2.6.4, the overhead induced by DC-net CT is a major challenge with regards to the realizability of DC-nets. We introduced ADC-nets in Section 4.4 to address this challenge. However, in Section 4.4.2, we also mentioned that the communication model of ADC-nets prevents collision detection. To address this challenge, we proposed a slightly modified version of the ADC-net communication model that can be used when collision detection is required.

We now compare the communication and computation overhead of classical DC-nets, ADC-nets and ADC-nets with collision detection. For both comparisons, we assume $p \geq 2$ as number of DC-net/ADC-net participants. Furthermore, for ADC-nets, we assume r as mean number of recipients per round to which a RP has to forward a published message.

Communication Overhead

Assume m as number of messages which are sent over the network for each round of communication. We now compare m for the different communication models and visualize the results in Table 6.2.

- *DC-nets*: Every participant has to send a DC-net message to each other participant.

- *ADC-nets*: Every participant, except the RP, sends its ADC-net message to the RP. The RP does not send its ADC-net message to itself over the network but just stores is locally. After decrypting the published secret message, the RP sends it to its r recipients.
- *ADC-nets with collision detection*: In contrast to regular ADC-nets, the RP also sends the published message to each other ADC-net participant for collision handling purposes.

Hereby, the amount of messages per round can be drastically reduced by using ADC-nets instead of DC-nets, especially for larger communication groups.

Computation Overhead

Assume o as number of XOR operations which are executed for each round of communication. We now compare o for the different communication models and visualize the results in Table 6.2.

- *DC-nets*: Every participant XORs its secret message with the round secrets it shares with each other participant. Furthermore, every participant XORs all received ADC-net messages of the round.
- *ADC-nets*: Every participant XORs its secret message with the round secrets it shares with each other participant. Only the RP XORs all received ADC-net messages of the round.

Hereby, the amount of XOR operations per round can be reasonably reduced by using ADC-nets instead of DC-nets. While RPs execute the same amount of XOR operations as each participant in a DC-net, the other ADC-net peers are relieved of decrypting a message for each round.

6.3 Group Setups

6.3.1 Approach

The group setup approach presented in Section 4.5.2 is designed to be comparably simple and therefore easy to implement. To allow the improvement of the approach in future work, we evaluate the impact of different group setup parameters on the following group setup properties:

- *Total Path Length*: the combined length of all random walks.
- *Visited Peers*: the number of peers visited by all random walks. Peers that are visited by different random walks are counted
  - (*total*) multiple times.
  - (*unique*) only once.
- *Participants*: the number of peers recruited by all random walks. Peers that are recruited by different random walks are counted
  - (*total*) multiple times.
  - (*unique*) only once.

Table 6.3.: The group setup parameter settings used for evaluating the impact of the respective parameter on the group setup properties presented in Section 6.3.1.

Parameter	Used Settings
Number of Walks	1, 3, 5
Group Size Target	10, 20, 50
Participation Probability	0.25, 0.5, 0.75, 0.99
Timeout	None, 5, 10, 25

We desire a minimal total path length to only induce a minimal communication overhead while achieving a certain number of participants. Therefore, the difference between visited peers and participants should be minimal. Furthermore, when executing multiple random walks in parallel, it is desirable that every peer/participant is only visited/recruited by one random walk to prevent unnecessary communication overhead.

6.3.2 Experimental Setup

We used the `group_setup_simulation` script, which is presented in Section 5.4.3, for the simulation of group setups with different configurations. We chose a number of settings for selected parameters of the `group_setup_simulation` script and list them in Table 6.3. These settings allow us to compare the impact of the parameters on the properties presented in Section 6.3.1.

We used a network with 1,000 peers and a degree of 4 for the simulations. This network was generated using the `network_generation` script presented in Section 5.4.2. Furthermore, the initiator of each group setup was randomly selected from all peers. We ran 10,000 simulations for each combination of the parameter settings listed in Table 6.3, resulting in 144 different combinations and 1,440,000 simulations in total.

6.3.3 Results

Our results are visualized as parallel coordinates in the Figures 6.1 to 6.4. For each Figure, we focus on one group setup parameter. This parameter is visualized in the first axis of the respective graph. The other axes show the group setup properties described in Section 6.3.1. For each selected setting of the respective group parameter, the properties of each group setup simulated for this parameter are averaged. For instance, each line in Figure 6.1 shows the average of $\frac{144}{3} \cdot 10,000$ simulations. In the following sections, we evaluate the impact of each parameter listed in Table 6.3 on the properties presented in Section 6.3.1.

Number of Walks

In Section 4.5.2, we proposed the use of multiple random walks as potential disruption countermeasure and as way for reducing the power of individual adversaries that try to control a considerable portion of the resulting group. We now evaluate the impact of using different numbers of random walks in parallel by reference to Figure 6.1.

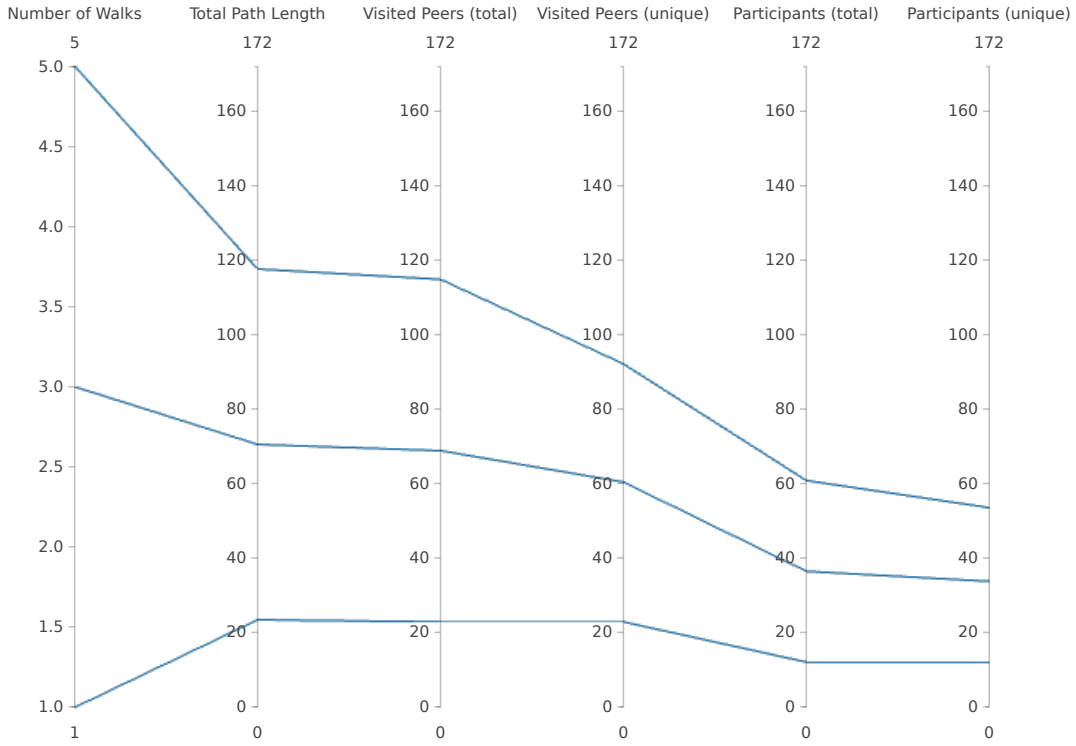


Figure 6.1.: The impact of using different number of walk settings on the group setup properties.

As expected, the best results are achieved when using a single random walk. Here, it is known at all times which peers have been visited and recruited. Hence, visited peers are only re-visited when backtracking.

With an increasing number of random walks, the number of peers being visited by multiple random walks also increases, hereby adding additional communication overhead. This is a result of the different random walks not being coordinated. However, for a complete lack of coordination, the percentage of peers/participants that are only visited/recruited by a single random walk is relatively high. We expect that the performance can be further improved by periodically exchanging information regarding the visited and recruited peers between the different random walks. While using multiple random walks is more complex, the opportunities presented in Section 4.5.2 may outweigh the additional effort.

Group Size Target

Due to its impact on anonymity set size, group size is a central aspect of our group setup approach. We now evaluate the impact of using different group size target settings by reference to Figure 6.2.

First, it can be seen that the respective group size target is exceeded for all settings. This is the result of using multiple random walks without coordination. As no knowledge regarding the recruited participants is shared between different random walks, each walk continues until either the desired group size is reached or the walk times out. Therefore, the resulting group size depends on all parameters listed in Table 6.3. They all have to be coordinated to achieve a specific group size.

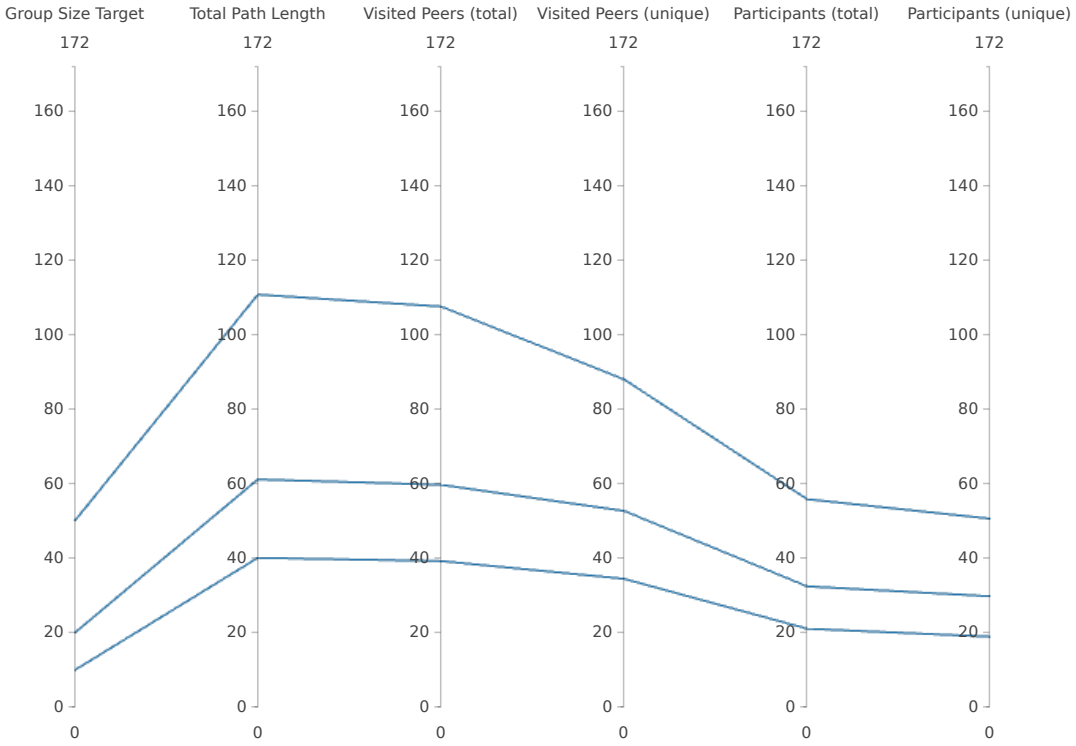


Figure 6.2.: The impact of using different group size target settings on the group setup properties.

Participation Probability

We introduced the concept of capacity in Section 4.5.1 and use it as central aspect of our group setup approach as presented in Section 4.5.2. For simplicity, we use the participation probability parameter instead of simulating the capacity for each peer in the network. We now evaluate the impact of using different participation probabilities by reference to Figure 6.3.

As expected, the participation probability has a direct effect on the ratio of visited peers to participants. The higher the participation probability, the more visited peers are also recruited as participants. This also explains the longer paths for lower participation probabilities as in average more peers have to be visited until the desired number of participants has been recruited. We also observed this effect for our implementation of the group setup technique presented in Section 4.5.2. As soon as a large majority of the peers had reached their capacity, the path length drastically increased.

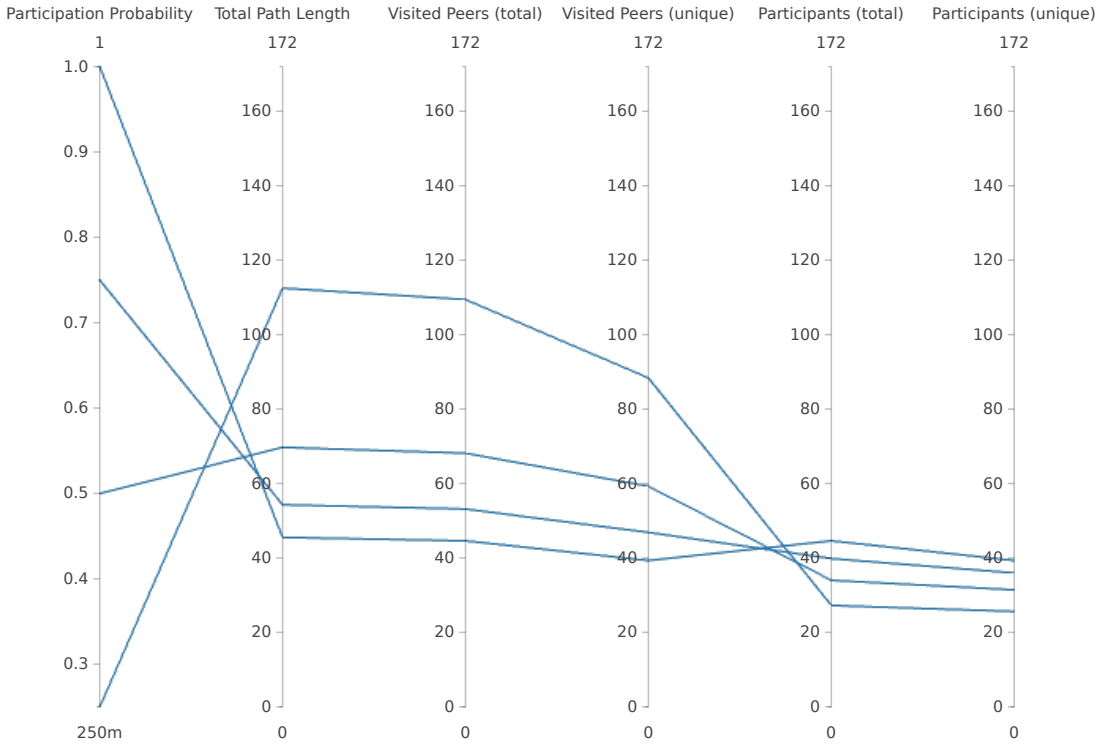


Figure 6.3.: The impact of using different participation probability settings on the group setup properties.

Timeout

In Section 4.5.2, we proposed the use of a timeout parameter as measure for preventing both the recruitment of too many participants and the traversal of the whole network in case of only few peers having capacities left. We now evaluate the impact of using different timeout settings by reference to Figure 6.4.

As can clearly be seen, using no timeout drastically increases the total path length. This is especially critical in scenarios where only very few peers have capacities left. Then, a large portion of network is traversed. In the worst case, the whole network is traversed without reaching the group size target, hence inducing a massive communication overhead without a new group being formed successfully. We also observed this effect for our implementation of the group setup technique presented in Section 4.5.2. Our results indicate that timeouts should always be used.

There is clearly no timeout setting which is perfect for all scenarios. Too low settings result in too few and too high settings in too many participants being recruited. Therefore, the timeout setting has to be selected with regards to the other group setup parameters.

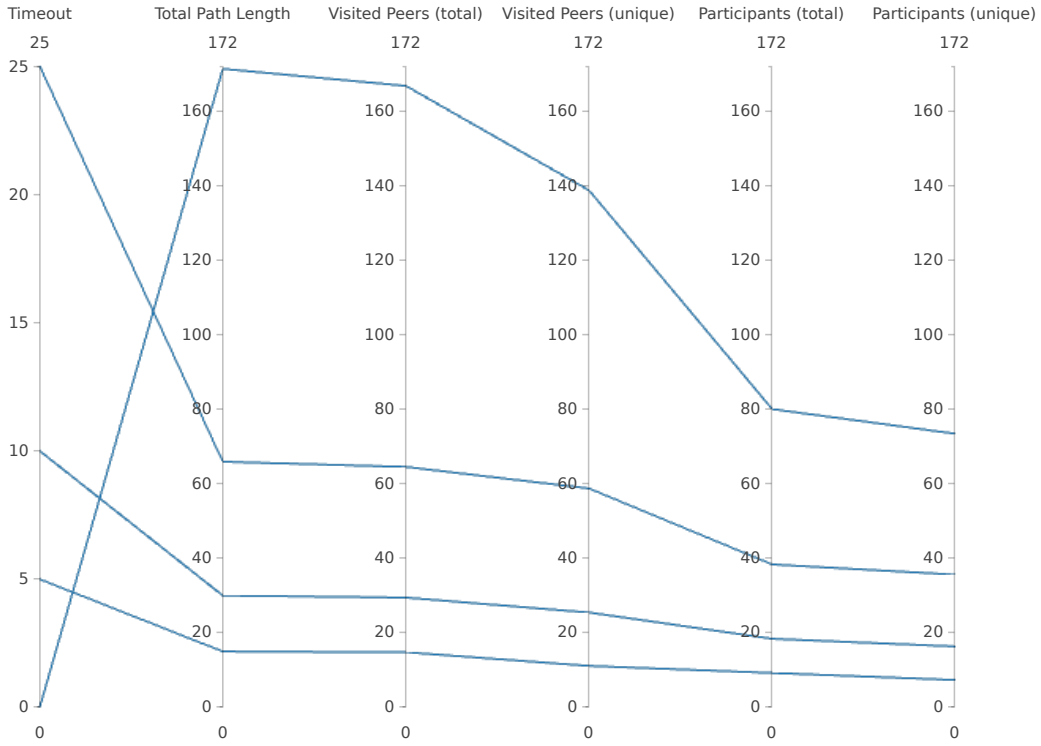


Figure 6.4.: The impact of using different timeout settings on the group setup properties. The *None* timeout setting is visualized as zero.

6.4 Rendezvous Point Elections

6.4.1 Approach

The RP election scheme presented in Section 4.4.1 is designed to randomly elect a leader from a set of candidates. We evaluate whether a number of elections in the same network results in a near-uniform distribution of leaders among all network peers. This is interesting in terms of load balancing. Additionally, we evaluate the speed of elections for different group sizes by comparing the following properties:

- *mean drop-out rate*: the mean percentage of candidates that are eliminated per election round.
- *number of rounds*: the number of election rounds until a candidate is elected as RP.

RPs should be elected as fast as possible to reduce the duration of ADC-net setups. Therefore, a high mean drop-out rate and a low number of rounds are desirable.

6.4.2 Experimental Setup

We used the `batch_evaluation` script, which is presented in Section 5.4.4, for running a number of comparable `adc_net` simulations for a selection of *candidate set sizes*.

Table 6.4.: Modified `adc_net` configuration options for running approximately 1,000 RP elections with $c \in \{10, 20, 50, 100\}$ candidates in a network with 100 peers.

Option	Used Setting
group.capacity	$10 \cdot c$
group.size.min	c
group.size.target	c
adc.enabled	false

As most of the `adc_net` configuration options are irrelevant for RP elections, we rely on the default settings listed in the Tables A.1, A.2 and A.3 unless mentioned otherwise. We chose $c \in \{10, 20, 50, 100\}$ as candidate set sizes. Furthermore, we decided to use networks with 100 peers for the simulation as this size is sufficient for the selected candidate set sizes. For each simulation a new network was generated using the `network_generation` script presented in Section 5.4.2. Apart from the number of network nodes, its default configuration, which is listed in Table A.5, was used.

We modified a number of `adc_net` settings so that approximately 1,000 elections are executed for each c. We can only approximate the number of elections as a number of groups may not be set up successfully during our simulation, resulting in less than 1,000 groups in which elections can be conducted. To reduce the simulation overhead, we disabled ADC-net functionality and therefore stop the ADC-net logic after RP elections. We list the modified `adc_net` settings in dependency to c in Table 6.4.

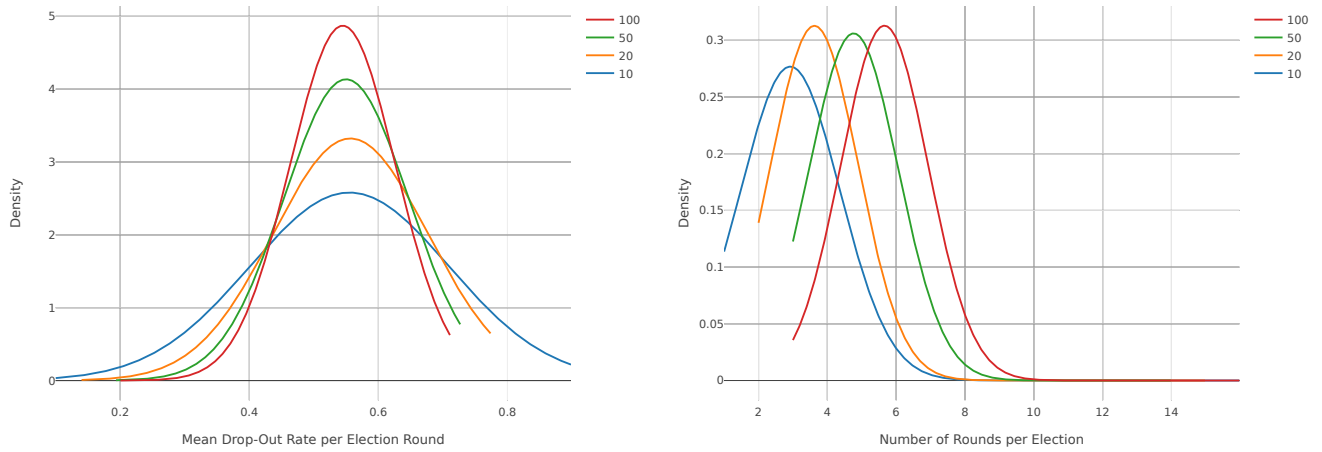
We ran each simulation for 150 seconds, resulting in 1,500 intervals at the default periodic interval of 0.1. This duration was sufficient for more than 90% of the 1,000 elections being conducted for each setting. Also, we executed 5 simulations for each c and averaged the results.

6.4.3 Results

Our results are visualized as distribution normal curves in the Figure 6.5 and as line plot in Figure 6.6. In the following sections, we evaluate the impact of the different candidate set sizes on the properties presented in Section 6.4.1.

Election Speed

Figure 6.5a clearly shows that the center of all mean drop-out rate distributions is located between 0.5 and 0.6. Hence, in average, more than half of the current candidates are eliminated per round. This behavior can also be observed in Figure 6.5b. While elections with more candidates in general take longer than elections with less candidates, the number of additional rounds is quite small when compared to the number of additional candidates.



(a) The distribution of mean drop-out rates for different candidate set sizes.

(b) The number of rounds per elections for different candidate set sizes.

Figure 6.5.: Both (a) and (b) show the fast speed of RP elections.

Leader Distribution

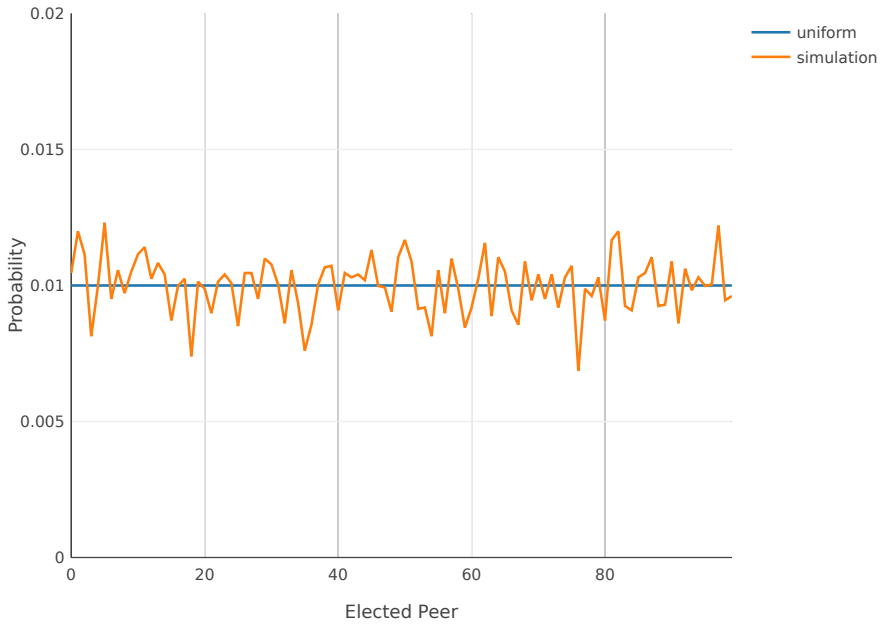


Figure 6.6.: Comparison of the uniform distribution and the distribution of leaders in the network averaged for 4,000 elections.

Figure 6.6 shows the distribution of leaders in the network. It can clearly be observed that the leader distribution approximates the uniform distribution. This indicates that the election scheme presented in Section 4.4.1 indeed elects each ADC-net participant with equal probability as RP.

Table 6.5.: Modified `adc_net` configuration for running the ADC-net collision handling evaluations for different wait probabilities $wp \in \{0.5, 0.75, 0.9\}$.

Option	Used Setting
group.init_prob	0.001
adc.collision_handling.wait_prob	*wp*
adc.transiency.enabled	false

Table 6.6.: The different sender control mode settings used for the ADC-net collision detection and ADC-net transiency simulations. Sender control modes are presented in Section 5.4.4.

	(sender rate, send probability)		
	(0.01, 0.05)	(0.1, 0.05)	(0.2, 0.05)
(0, 0)	(0.01, 0.2)	(0.1, 0.2)	(0.2, 0.2)
	(0.01, 0.5)	(0.1, 0.5)	(0.2, 0.5)

6.5 ADC-net Collision Handling

6.5.1 Approach

As presented in Section 5.3.7, we implemented a straightforward collision handling technique to allow the use of multiple senders per ADC-net. While we do not focus on collision handling in the theoretical part of this thesis, we still evaluate the impact of different sender control modes, which are presented in Section 5.4.4, on the following aspects:

- the *percentage of non-empty messages* that are published in the ADC-net.
- the *percentage of collisions for* these *non-empty messages*.
- the *mean delay* of *messages* that are *tracked* using the `sender_control` message tracking functionality presented in Section 5.3.7.

6.5.2 Experimental Setup

We used the `batch_evaluation` script, which is presented in Section 5.4.4, for running a number of comparable `adc_net` simulations for different sender and collision handling configurations. For each simulation, a new network with 1,000 peers was generated using the `network_generation` script presented in Section 5.4.2 with its default configuration as listed in Table A.5.

As in Section 6.4.2, we only modified few of the `adc_net` configuration options and relied on the default settings listed in the Tables A.1, A.2 and A.3 for the remaining options. We used a group setup initialization probability of 0.001. Hereby, in average, one of the 1,000 peers initiates a group setup for a new ADC-net per periodic interval. Due to the default group capacity and group size settings, exactly $\frac{1000}{25} = 40$ ADC-nets, each comprising 25 peers, are set up for

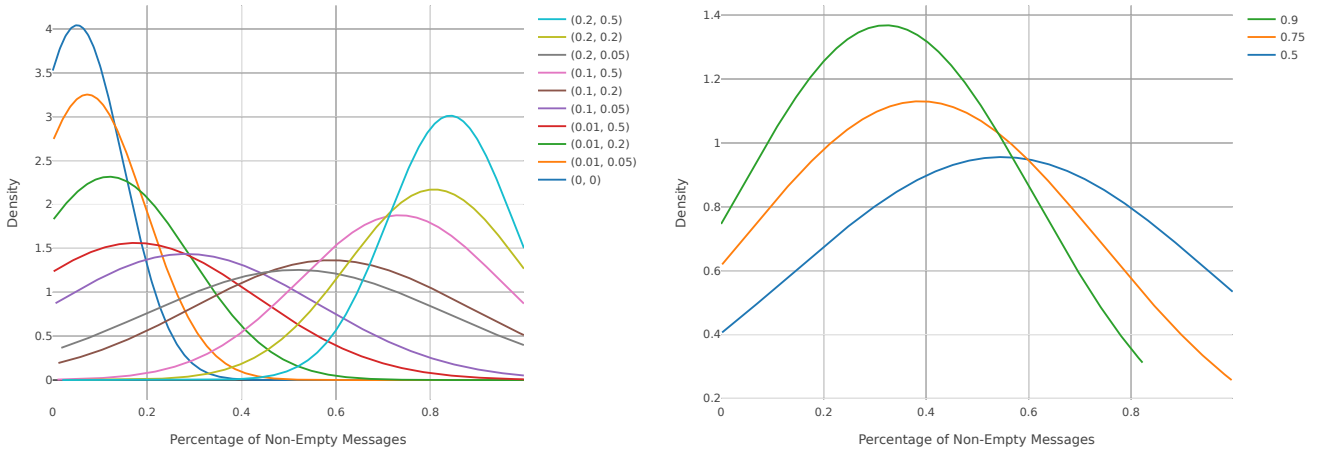
every simulation. With the specified group setup initialization probability, these 40 ADC-nets are quickly set up. Hence, they all run over a comparable number of round, allowing us to compare their properties. With transient behavior, some ADC-nets are not running continuously, hence impacting the mentioned comparability. Therefore, we disabled transient ADC-net behavior.

To evaluate the impact of different numbers of senders and different numbers of messages sent by each sender, we used the `batch_execution` sender control modes listed in Table 6.6.

We ran each simulation for 500 seconds, resulting in 5,000 intervals at the default periodic interval of 0.1. With a default ADC-net interval coefficient of 10, up to 500 rounds of ADC-net communication could be observed for this configuration. We executed 3 simulations for each combination of the sender and the collision handling configurations, resulting in 30 different combinations and 90 simulations in total. We averaged the results for each configuration combination over the 3 simulations executed for this combination.

6.5.3 Results

Percentage of Non-Empty Messages

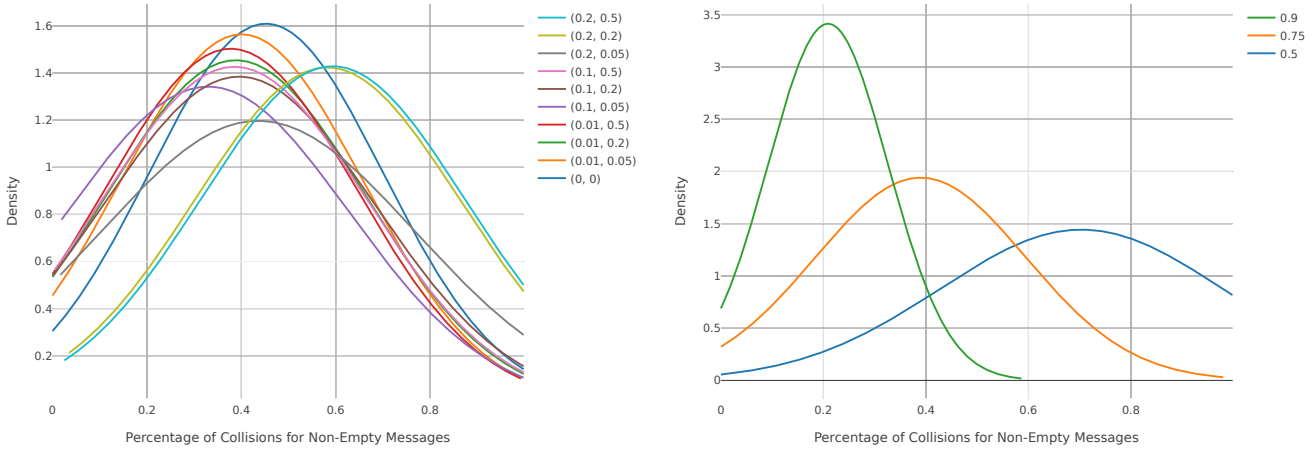


(a) Different Sender Modes **(b)** Different Collision Wait Probabilities

Figure 6.7.: The percentage of non-empty messages increases with higher sender rates, higher send probabilities and lower collision wait probabilities.

As expected and observed in Figure 6.7a, the number of non-empty messages being published in ADC-nets increases with a higher sender rate and a higher send probability. Furthermore, it can be observed that non-empty secret messages are also published if the sender rate is zero. This is due to the use of interest control messages as presented in Section 4.5.3.

In Figure 6.7b, it is shown that higher wait probabilities result in fewer non-empty secret messages being published in ADC-nets. This is expectable as the average wait time of each peer that was part of a collision in an ADC-net is increased. Hereby, the number of non-empty secret messages it can publish in the same ADC-net is reduced.



(a) Different Sender Modes

(b) Different Collision Wait Probabilities

Figure 6.8.: The percentage of collisions for non-empty messages increases with higher sender rates, higher send probabilities and smaller collision wait probabilities.

Percentage of Collisions for Non-Empty Messages

In Figure 6.8a, it can clearly be observed that an increase in sender rate and send probability increases the probability of published non-empty messages being part of a collision. Furthermore, the similarity of the results for the same sender rates and send probabilities of 0.2 and 0.5 should be noted. This indicates that a send probability of 0.2 is already too high as it results in the same number of collisions as the much higher setting of 0.5.

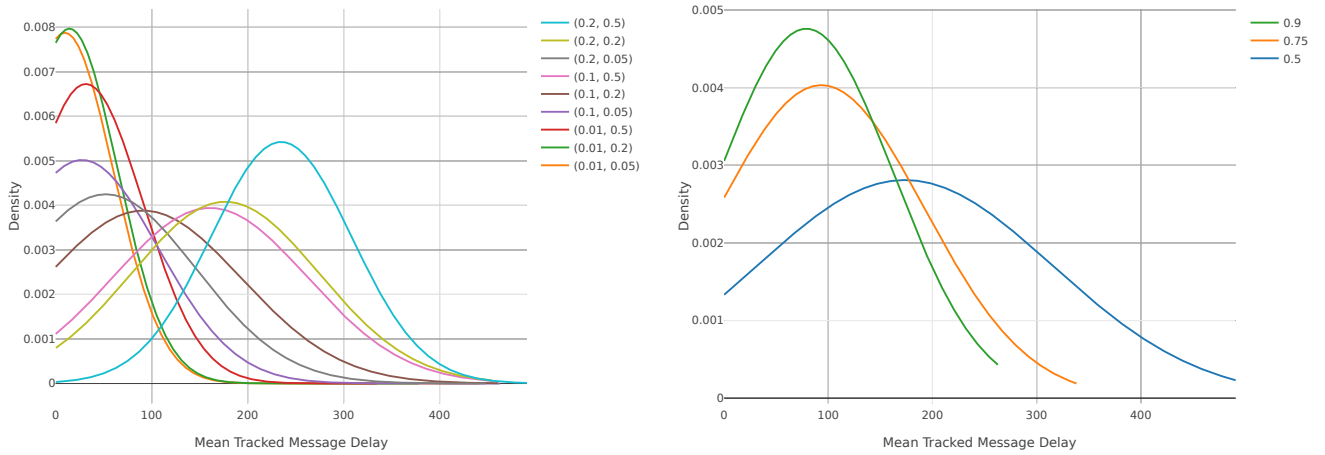
In the last section, we showed that higher wait probabilities result in fewer non-empty secret messages being published in ADC-nets. Now, we can observe in Figure 6.8b that high wait probabilities result in a considerable reduction in the number of collisions when compared to smaller probabilities. Hence, while more messages can be published with a smaller wait probability, this is not necessarily positive as the majority of these messages is part of a collision and therefore has to be sent again.

Mean Tracked Message Delay

Depending on the number of collisions in an ADC-net, secret messages may have to be sent several times before they do not collide with other secret messages and thereby are successfully published. This has a severe impact on the message delay as each peer has to wait for a certain time before retransmitting secret messages that previously collided.

In the last section, we saw an increasing number of collisions for high sender rates and high send probabilities. This also has a direct impact on the message delay as can clearly be observed in Figure 6.9a. The same applies to our previous findings with regards to the number of collisions for different wait probabilities and is displayed in Figure 6.9b.

Our results indicate that higher wait probabilities in general have a positive effect on the number of non-empty secret messages that are successfully published in ADC-nets. Yet, we expect that too high wait probabilities are counterproductive, especially with regards to message



(a) Different Sender Modes **(b)** Different Collision Wait Probabilities

Figure 6.9.: Mean Tracked Message Delay

Figure 6.10.: The mean tracked message delay increases significantly with higher sender rates, higher send probabilities and smaller collision wait probabilities.

delay. Therefore, it may be interesting to analyze the behavior over a longer period of time. We leave this task to future work.

6.6 ADC-net Transiency

6.6.1 Approach

We presented transient behavior as a technique for reducing the continuous overhead induced by ADC-nets in Section 4.4.3. We now analyze the impact of different sender modes and threshold settings on the time until ADC-nets are disbanded. We measure this time by counting the number of ADC-net communication rounds in the respective ADC-net until the transiency transition was initiated and refer to it as *transition start round*.

6.6.2 Experimental Setup

Table 6.7.: Modified `adc_net` configuration options for running ADC-net transiency simulations for different transiency transition thresholds $tt \in \{10, 30, 50\}$.

Option	Used Setting
group.init_prob	0.001
adc.transiency.transition.init_prob	1
adc.transiency.transition.threshold	tt

The experimental setup used for the ADC-net transiency simulations is very similar to the one presented in Section 6.5.2. Therefore, we only present the differences to the previously presented experimental setup:

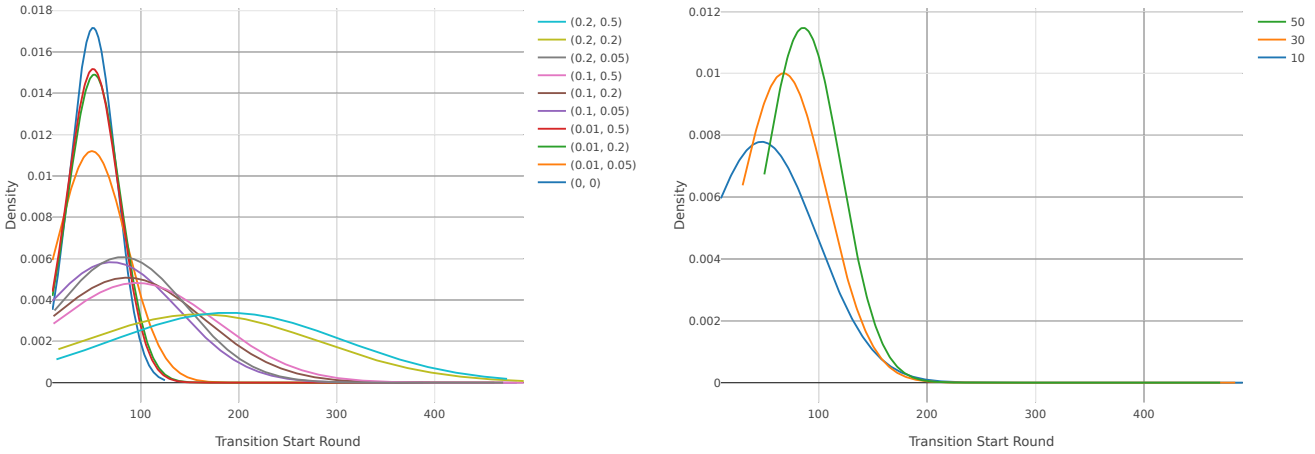
- Transient ADC-net behavior was enabled.
- We used different ADC-net transiency transition thresholds instead of different ADC-net wait probabilities.
- We used the default ADC-net wait probability as listed in Table A.3.

Furthermore, we also used 0.001 as group setup initialization probability so that both

1. the first 40 ADC-nets and
2. the ADC-nets that are created after one of the initial 40 ADC-nets has been disbanded

are quickly set up. This way, we can observe the behavior of a high number of ADC-nets in a relatively short time span.

6.6.3 Results



(a) Different Sender Modes **(b)** Different Transition Thresholds

Figure 6.11.: The transition start rounds increase significantly with higher sender rates and send probabilities and reasonably for higher transition thresholds.

In Figure 6.11a, we can observe that the transition start round increases for higher sender rates and send probabilities. This was expected as our transiency approach disbands ADC-nets when only empty messages have been published for a number of consecutive rounds.

It should be noted that the orange curve for the sender rate of 0.01 and the send probability of 0.05 is an outlier. We expected it to be located between the blue and the green curve. However, after analyzing the evaluation data for the simulations conducted with this setting and comparing them to the other simulations, we can explain its occurrence: At a sender rate of 0.01 and a send probability of 0.05, more ADC-nets were disbanded for higher transition threshold settings than for higher sender rates and send probabilities. Therefore, the center of the transition start round distribution for this setting is not as dense as expected. This outlier shows that the different sender rates and send probabilities are only comparable to a certain extent. Still, Figure 6.11a clearly indicates that sender rates and send probabilities have a significant impact on the transition start rounds.

In Figure 6.11b, it can be observed that higher transition thresholds increase the mean transition start round as we expected. Furthermore, we see that most of the transitions happen comparably early. Yet, this may also be linked to the outlier problem mentioned before and has to be evaluated further by running simulations over a longer timespan. We leave this task to future work.

6.7 Research Questions

When we started with the research associated with this thesis, we formulated a number of research questions which we wanted to address. We can now answer these questions and do so in the following sections.

6.7.1 How to create the overlay without jeopardizing the identity of a sender?

In Section 4.5, we showed that RPs of ADC-nets can be used as proxies for overlay interactions. Using this approach, senders cannot be linked to overlay initializations as they never directly interact with overlays in their role as sender. Furthermore, we presented a technique for the randomized periodic initialization of ADC-nets in Section 4.5.1. This technique prevents that a sender intentionally, in its role as sender, initiates an ADC-net. This is important as otherwise, adversaries could identify the initiator of an ADC-net as initiator of an overlay that is initialized using the RP of the ADC-net as proxy.

6.7.2 How long does it take until an overlay for a set of interests is set up?

The following tasks have to be carried out before an overlay can be set up:

1. *ADC-net Initialization*: The ADC-net setup has to be randomly initiated by a peer, for instance, using the approach presented in Section 4.5.1.
2. *Group Setup:* A number of participants for the new ADC-net have to be found, for instance, using the approach presented in Section 4.5.2.
3. *RP Election:* A RP has to be elected for the ADC-net, for instance, using the approach presented in Section 4.4.1.
4. *ADC-net Cryptography Setup:* Pair-wise master secrets have to be agreed upon as presented in Section 2.6.1.

For our presented techniques, the group setup phase most likely has the longest duration due to the distributed traversal of the network. The election phase instead is comparably short due to the fast speed of the used election scheme as showed in Section 6.4. How long each phase takes in a real-world environment is hard to estimate and depends on the specific implementation.

6.7.3 Is it always possible to create ADC-net overlays with a previously set number of participants or is only a range of acceptable sizes practically feasible?

In Section 6.3, we showed that a previously set number of participants can be problematic to achieve, especially when only few peers have capacities left for taking part in a new ADC-net.

Here, a range of acceptable group sizes should be used in combination with a group setup timeout. This prevents the traversal of large portions of the network which would induce a considerable communication overhead, especially for large networks. When assuming a network in which most of the peers have enough capacities left for participating in a new ADC-net, a previously set number of participants is also feasible.

6.7.4 How does the anonymity set size of ADC-nets compare to that of DC-nets?

Classical DC-nets comprise all peers and hence exhibit the maximum anonymity set size that can be achieved in a network. ADC-nets in turn comprise a configurable number of peers, when used as presented in Section 4.5, and therefore exhibit configurable anonymity set sizes. Also, the anonymity set size of ADC-nets only applies to sender operations as they, in contrast to classical DC-nets, do not protect receiver anonymity. Therefore, they can only be used for sender protection.

A configurable anonymity set size is desirable as it allows the selection of different anonymity set sizes for different use cases. For instance, communication with critical content can be protected by using larger anonymity sets at the cost of increased overhead.

6.7.5 How does the communication overhead of ADC-nets compare to that of DC-nets?

As presented in Section 6.2.2, ADC-nets induce significantly less communication overhead than classical DC-nets, especially for a high number of DC-net/ADC-net participants. We summarize our results assuming $p \geq 2$ as number of DC-net/ADC-net participants, r as mean number of recipients per round to which a RP has to forward a published message and m as the number of messages sent for each round of communication:

- *DC-nets*: Every participant has to send a DC-net message to each other participant.

$$m = p \cdot (p-1)$$

- *ADC-nets*: Every participant, except the RP, sends its ADC-net message to the RP. After decrypting the published secret message, the RP sends it to its r recipients.

$$m = (p-1) + r$$

- *ADC-nets with collision detection*: In contrast to regular ADC-nets, the RP also sends the published message to each other ADC-net participant for collision handling purposes.

$$m = 2 \cdot (p-1) + r$$

Our results show that ADC-nets can be used as more efficient sender protection component than classical DC-nets. While ADC-nets with collision detection exhibit a higher communication overhead than ADC-nets without collision detection, they still perform much better than classical DC-nets.

6.7.6 How does the computational overhead of ADC-nets compare to that of DC-nets?

As presented in Section 6.2.2, ADC-nets induce considerably less computation overhead than classical DC-nets. We summarize our results assuming $p \geq 2$ as number of DC-net/ADC-net participants, and o as the number of XOR operations executed for each round of communication:

- *DC-nets*: Every participant XORs its secret message with the round secrets it shares with each other participant. Furthermore, every participant XORs all received ADC-net messages of the round.

$$o = p \cdot (p-1) + p \cdot (p-1)$$

- *ADC-nets*: Every participant XORs its secret message with the round secrets it shares with each other participant. Only the RP XORs all received ADC-net messages of the round.

$$o = p \cdot (p-1) + p - 1$$

ADC-nets exhibit considerably reduced computational overhead when compared to classical DC-nets. Each ADC-net participant, except the RP, only has to execute $p-1$ XOR operations while the RP has to execute $2 \cdot (p-1)$ XOR operations which is the same as each participant of a classical DC-net.

7 Conclusion

7.1 Motivation

For privacy-friendly communication, the content of messages has to be protected. However, the content of messages is not the only aspect of communication that is relevant for privacy. It can already be critical if an adversary can identify the participants of a communication. Hence, techniques for protecting the identity of communication participants are required. While a number of feasible techniques exist for one-to-one communication, the ones that can be used for many-to-many communication induce a significant overhead and hence are not scalable. This is critical as many widespread technologies, e.g., social networks and the IoT, depend on many-to-many communication.

7.2 Contributions

We presented a number of challenges that are associated with sender protection in P2P networks when using pub-sub overlays for scalable many-to-many communication. We introduced ADC-nets as modified version of classical DC-nets with reduced overhead and maintained sender anonymity guarantees. We presented a novel approach that uses intermediate ADC-nets with configurable group sizes for sender protection. As part of this approach, we introduced a technique for forming suitable anonymization groups for a range of acceptable group sizes.

7.3 Results

We showed that ADC-nets, when compared to classical DC-nets with the same number of participants, considerably reduce the overhead induced by CT while maintaining the anonymity set size for send operations. As such, they can be used as more efficient component of sender protection techniques than classical DC-nets.

Our approach prevents the leakage of sender-related information during

- the setup of ADC-nets for sender protection,
- sender-related overlay interactions and
- the sending of messages.

Hereby, adversaries cannot identify senders within the anonymity set of the respective ADC-net.

In addition to our group forming technique, we conducted an extensive evaluation regarding the impact of several group setup parameters on selected group setup properties. Our results indicate that desired group sizes can be achieved by closely coordinating the analyzed parameters. We also show that a timeout should always be used for group setups to prevent the traversal of large portions of the network and hereby a considerable communication overhead.

7.4 Future Work

We now present a number of aspects that could be addressed in future work:

- *Group Setup Coordination:* We showed that the use of multiple random walks may increase the disruption-resilience of group setups and also could restrict adversaries from controlling a large portion of the resulting group. However, we also showed that using multiple random walks can be problematic for larger groups when used without any coordination. We expect that the use of periodic coordination between multiple random walks can improve the performance of groups setups by preventing visiting/recruiting a peer multiple times. Therefore, it may be interesting to evaluate how periodic coordination can be realized and whether it has the expected effect.

- *ADC-net Pausing and Disbanding:* We presented pausing and disbanding of ADC-nets as option to further reduce overhead. It may be interesting to evaluate the impact of different pausing durations and disbanding thresholds on message delay and compare it to the achieved overhead reduction. The results could then indicate which pausing durations and disbanding thresholds are suitable for different use cases.

- *Disruption Resilience:* Disruption is a major problem that has to be addressed before our approach can be used in a real-world application. It may be interesting to think about ways to mitigate disruption by adversarial RPs, for instance, by periodically electing new RPs in ADC-nets.

Acronyms

ADC-net Asymmetric Dining Cryptographers network

CLI command-line interface

CSPRNG cryptographically secure pseudo-random number generator

CT cover traffic

DC-net Dining Cryptographers network

DH Diffie-Hellman key exchange

ECC elliptic-curve cryptography

ECDH elliptic-curve Diffie-Hellman

GIL global interpreter lock

IoT Internet of Things

KDF key derivation function

LB Lightest Bin

MAC medium access control

MITM man-in-the-middle

Mix net Mix network

OTP one-time pad

P2P peer-to-peer

PKI public key infrastructure

pub-sub publish-subscribe

RP rendezvous point

List of Figures

List of Tables

Bibliography

[1] Openssl speed documentation. https://www.openssl.org/docs/manmaster/man1/speed.html (Accessed: 27.09.2018).

[2] Python global interpreter lock. https://wiki.python.org/moin/GlobalInterpreterLock (Accessed: 18.09.2018).

[3] Dennis Andriesse, Christian Rossow, Brett Stone-Gross, Daniel Plohmann, and Herbert Bos. Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus. In *2013 8th International Conference on Malicious and Unwanted Software:*, pages 116–123. IEEE, 2013.

[4] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 11–20. ACM, 2007.

[5] Michael Ben-Or and Nathan Linial. Collective coin flipping. *Advances in Computing Research*, 5:91–115, 1989.

[6] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web mixes: A system for anonymous and unobservable internet access. In *Designing privacy enhancing technologies*, pages 115–129. Springer, 2001.

[7] Owen Bowcott. Gchq data collection regime violated human rights, court rules, Sep 2018. https://www.theguardian.com/uk-news/2018/sep/13/gchq-data-collection-violated-human-rights-strasbourg-court-rules (Accessed: 23.09.2018).

[8] John W Byers, Michael Luby, and Michael Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected areas in Communications*, 20(8):1528–1540, 2002.

[9] Carole Cadwalladr and Emma Graham-Harrison. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach, Mar 2018. https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election (Accessed: 11.09.2018).

[10] Duncan Campbell. Inside echelon, Jul 2000. https://www.heise.de/tp/features/Inside-Echelon-3447440.html (Accessed: 11.09.2018).

[11] Jo-Mei Chang and Nicholas F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems (TOCS)*, 2(3):251–273, 1984.

[12] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[13] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[14] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 340–350. ACM, 2010.

[15] Jörg Daubert, Mathias Fischer, Stefan Schiffner, and Max Mühlhäuser. Distributed and anonymous publish-subscribe. In *International Conference on Network and System Security*, pages 685–691. Springer, 2013.

[16] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[17] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131, 2003.

[18] Uriel Feige. Noncryptographic selection protocols. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 142–152. IEEE, 1999.

[19] Sally Floyd, Van Jacobson, C-G Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM transactions on networking*, 5(6):784–803, 1997.

[20] Sean Gallagher. Equifax breach exposed millions of driver's licenses, phone numbers, emails, May 2018. `https://arstechnica.com/information-technology/2018/05/equifax-breach-exposed-millions-of-drivers-licenses-phone-numbers-emails` (Accessed: 11.09.2018).

[21] Hector Garcia-Molina. Elections in a distributed computing system. *IEEE transactions on Computers*, (1):48–59, 1982.

[22] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *IEEE INFOCOM*, volume 1, pages 120–130. Citeseer, 2004.

[23] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[24] Philippe Golle and Ari Juels. Dining cryptographers revisited. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 456–473. Springer, 2004.

[25] Tim Grube, Sascha Hauke, Jörg Daubert, and Max Mühlhäuser. Ant colonies for efficient and anonymous group communication systems. In *Networked Systems (NetSys), 2017 International Conference on*, pages 1–8. IEEE, 2017.

[26] Tim Grube, Markus Thummerer, Jörg Daubert, and Max Mühlhäuser. Cover traffic: A trade of anonymity and efficiency. In *International Workshop on Security and Trust Management*, pages 213–223. Springer, 2017.

[27] Justin P Johnson. Targeted advertising and advertising avoidance. *The RAND Journal of Economics*, 44(1):128–144, 2013.

[28] M Frans Kaashoek, Andrew S. Tanenbaum, and Susan Flynn Hummel. An efficient reliable broadcast protocol. *ACM SIGOPS Operating Systems Review*, 23(4):5–19, 1989.

[29] Bruce M Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *ACM Transactions on Algorithms (TALG)*, 6(4):68, 2010.

[30] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 990–999. Society for Industrial and Applied Mathematics, 2006.

[31] Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In *Annual Cryptology Conference*, pages 631–648. Springer, 2010.

[32] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[33] Kristin Lauter. The advantages of elliptic curve cryptography for wireless security. *IEEE Wireless communications*, 11(1):62–67, 2004.

[34] Brian N Levine, Michael K Reiter, Chenxi Wang, and Matthew Wright. Timing attacks in low-latency mix systems. In *International Conference on Financial Cryptography*, pages 251–265. Springer, 2004.

[35] Ewen Macaskill, Gabriel Dance, Feilding Cage, and Greg Chen. Nsa files decoded: Edward snowden's surveillance revelations explained, Nov 2013. `https://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded` (Accessed: 11.09.2018).

[36] Joseph Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations. Jan 1999. `https://tools.ietf.org/html/rfc2501.html`, (Accessed: 14.09.2018).

[37] Nayantara Mallesh and Matthew Wright. Countering statistical disclosure with receiver-bound cover traffic. In *European Symposium On Research In Computer Security*, pages 547–562. Springer, 2007.

[38] Sanjoy Paul, Krishan K. Sabnani, JC-H Lin, and Supratik Bhattacharyya. Reliable multicast transport protocol (rmtp). *IEEE journal on selected areas in communications*, 15(3):407–421, 1997.

[39] Nicole Perlroth. All 3 billion yahoo accounts were affected by 2013 attack. *New York Times*, Mar 2017. `https://www.nytimes.com/2017/10/03/technology/yahoo-hack-3-billion-users.html` (Accessed: 13.09.2018).

[40] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. 2010.

[41] Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM transactions on information and system security (TISSEC)*, 1(1):66–92, 1998.

[42] Lawrence G Roberts. Aloha packet system with and without slots and capture. *ACM SIGCOMM Computer Communication Review*, 5(2):28–42, 1975.

[43] Matthew Rosenberg, Nicholas Confessore, and Carole Cadwalladr. How trump consultants exploited the facebook data of millions, Mar 2018. https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html (Accessed: 11.09.2018).

[44] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[45] Elizabeth Stoycheff. Under surveillance: Examining facebook's spiral of silence effects in the wake of nsa internet monitoring. *Journalism & Mass Communication Quarterly*, 93(2):296–311, 2016.

[46] Paul Syverson, R Dingledine, and N Mathewson. Tor: The secondgeneration onion router. In *Usenix Security*, 2004.

[47] Michael Waidner, Birgit Pfitzmann, et al. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. *J.-J. Quisquater and J. Vandewalle, editors, Advances in Cryptology—EUROCRYPT*, 89:690, 1989.

[48] Brent R Waters, Edward W Felten, and Amit Sahai. Receiver anonymity via incomparable public keys. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 112–121. ACM, 2003.

[49] Alan F Westin and Oscar M Ruebhausen. *Privacy and freedom*, volume 1. Atheneum New York, 1967.

[50] Zack Whittaker. A new data leak hits aadhaar, india's national id database, Mar 2018. https://www.zdnet.com/article/another-data-leak-hits-india-aadhaar-biometric-database (Accessed: 11.09.2018).

Appendices

A Implementation Configuration Options

Table A.1.: Selection of `adc_net` main configuration options.

Option	Description	Type	Default
mode	The used main mode.	`str`	“simulation”
port	The port used by all peers for the communication.	`int`	9000
periodic_interval	The interval in seconds in which periodic tasks are executed.	`float`	0.1
group	Group Setup Settings, see Table A.2.	`object`	
adc	ADC-net Settings, see Table A.3.	`object`	

Table A.2.: The `adc_net` main configuration options regarding group setups.

Option	Description	Type	Default
capacity	The number of groups a peer can participate in simultaneously.	`int`	1
init_prob	The probability of a peer to initiate a group setup if its capacity is not yet reached.	`float`	0.01
size.min	The minimum number of participants for a group that are required for a setup to succeed.	`int`	25
size.target	The desired number of participants for group setups.	`int`	25

Table A.3.: The `adc_net` main configuration options regarding ADC-nets.

Option	Description	Type	Default
enabled	Whether ADC-net functionality is enabled. If disabled, only group setups and RP elections are conducted.	`bool`	true
checksum	Whether checksums are used for detecting collisions that do not result in a malformed JSON format. Required for detecting interest collisions at the RP.	`bool`	true
interval_coefficient	The number of periodic intervals to wait between ADC-net rounds.	`int`	10
msg_length	The length of the ADC-net round messages.	`int`	128
collision_handling.enabled	Whether collision handling is enabled.	`bool`	true
collision_handling.wait_prob	The probability which is checked every ADC-net round by peers who were part of a collision in the respective ADC-net and hence currently are in a wait period. This period continues until the continuously repeated probability check fails once.	`float`	0.9
transiency.enabled	Whether transient behavior is enabled.	`bool`	true
transiency.transition.duration	The duration of the transiency transition phase in ADC-net rounds.	`int`	10
transiency.transition.init_prob	The probability with which the transition is initiated. It is checked each round in which the threshold is reached.	`float`	0.5
transiency.transition.threshold	The threshold that has to be reached before the ADC-net can be disbanded. Counted in number of consecutive rounds in which no non-empty ADC-net message has to be sent over the ADC-net.	`int`	30

Table A.4.: The `adc_net` peer configuration options.

Option	Description	Type
peer.addr	The IP address of the communication interface used for communication with other peers.	`str`
peer.neigh	The IP addresses of adjacent neighbor peers.	`list of str`
peer.interests	The topics in which the peer is interested.	`list of str`

Table A.5.: The configuration options for the `network_generation` script.

Option	Description	Type	Default
nodes	The number of generated network nodes.	`int`	1000
degree	The mean node degree.	`float`	4
mode	The configuration export mode.	`str`	"simulation"
network	The network which is used for address allocation for the nodes.	`str`	"127.0.0.0/8"
interests_list	The list of potential interests.	`list of str`	["a", "b", ..., "z"]
interests_min	The minimum number of interests added to each node.	`int`	0
interests_add_prob	The probability for adding more interests when the minimum number of interests is reached. Interest are added subsequently until the check fails.	`float`	0.25

Table A.6.: The configuration options for the `sender_control` script.

Option	Description	Type
addr	The address of the addressed peer.	`str`
port	The port of the addressed peer.	`int`
interval	The interval in which the addressed peer appends a message to the send queue if the `msg_add_prob` check succeeds. This setting is used as coefficient of the `periodic_interval` parameter from the `adc_net` main configuration.	`int`
msg_add_prob	The probability with which a message is added to the send queue. It is checked each interval.	`float`
msg_number	The number of messages that are appended to the send queue before the sender control task is finished.	`int`
msg_candidates	A list of potential messages. Each message that is appended to the send queue is randomly selected from this list.	`list of str`
msg_tracking	Whether tracking information is added to send messages. This information is used for the evaluation to determine the delay of messages.	flag