Footprint scheduling for Dining-Cryptographer networks

Anna Krasnova[1], Moritz Neikes[1], and Peter Schwabe[1] *

Digital Security Group, Radboud University
Toernooiveld 212, 6525 EC Nijmegen, The Netherlands
anna@mechanical-mind.org, m.neikes@student.ru.nl, peter@cryptojedi.org

Abstract. In many communication scenarios it is not sufficient to protect only the content of the communication, it is necessary to also protect the identity of communicating parties. Various protocols and technologies have been proposed to offer such protection, for example, anonymous proxies, mix-networks, or onion routing. The protocol that offers the strongest anonymity guarantees, namely unconditional sender and recipient untraceability, is the Dining Cryptographer (DC) protocol proposed by Chaum in 1988. Unfortunately the strong anonymity guarantees come at the price of limited performance and scalability and multiple issues that make deployment complicated in practice.
In this paper we address one of those issues, namely slot reservation. We propose *footprint scheduling* as a new technique for participants to negotiate communication slots without losing anonymity and at the same time hiding the number of actively sending users. Footprint scheduling is at the same time simple, efficient and yields excellent results, in particular in very dynamic networks with a frequently changing set of participants and frequently changing activity rate.

Keywords: DC-net, scheduling, anonymity, slot-reservation

1 Introduction

"We kill people based on metadata" – this statement by former CIA and NSA director Michael Hayden demonstrates more than clearly that cryptographically protecting only the content of communication is insufficient; secure communication also has to protect the identities of the communicating parties. Various protocols and techniques have been proposed to enable anonymous communication. All of these techniques have to choose a trade-off between efficiency in terms of throughput, latency, and scalability on the one hand and security guarantees on the other hand. For example, anonymous proxies provide low latency and potentially very good throughput and scalability, but all participants' identities

* This research was conducted within the Privacy and Identity Lab (PI.lab, http://www.pilab.nl) and funded by SIDN.nl (http://www.sidn.nl/) and by Netherlands Organisation for Scientific Research (NWO) through Veni 2013 project 13114. Permanent ID of this document: 215ad4d1ccbd4ee7a6c763de5d2a8537. Date: December 18, 2015.

are compromised if one trusted node, the proxy, is compromised. Stronger guarantees are offered by a cascade of proxies in onion-routing networks like Tor [17], which were originally proposed by Syverson, Goldschlag, and Reed in [16]. However, onion-routing networks are possibly susceptible to attacks that correlate traffic entering and leaving the network. See, for example, [11]. Mix-nets, proposed already in 1981 by Chaum in [4] do not have this problem; however, they suffer from increased latency.

The anonymity protocol that offers the strongest guarantees is the *Dining-Cryptographers protocol*, also known as Dining-Cryptographers network or short DC-net, which was introduced by Chaum in [3]. The protocol provides unconditional communication anonymity for senders and recipients, however, at the cost of low throughput and high latency, in particular when scaling to many participants.

Dining cryptographers. To explain how DC-net works, consider an example with 3 participants exchanging 6-bit messages. Figure 1 depicts the example. Each participant has a shared symmetric key with each other participant. Assume that participant A wants to send a message. To do so, she xors her message with all the keys she shares with the other participants. The result is an *output* that A sends out to every participant of the DC-net. The other participants perform the same procedure, but use a zero message instead of a meaningful one. All outputs of all participants are xored together to reveal the meaningful message of the participant A, because keys are canceling out (since each symmetric key is used twice). This completes a single *round* of the DC-net, during which one participant can transfer (broadcast) one message; in the next round another participant transmits a message until all participants are done transmitting.

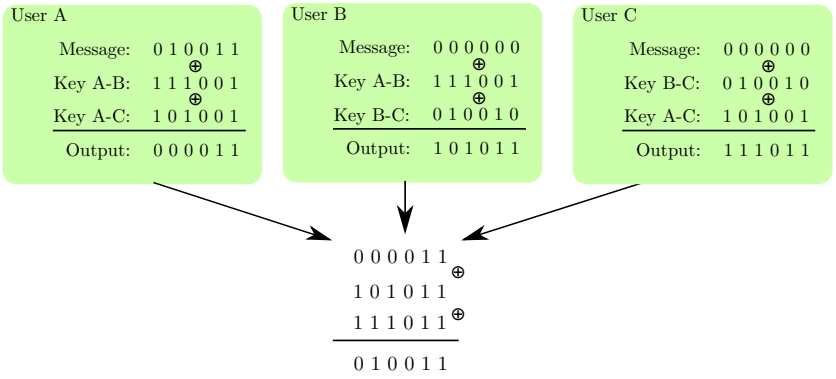

Fig. 1: DC-net with tree participants and 6-bit messages

Collisions and scheduling. If two participants send in the same round, their messages collide and become unusable. This can happen accidentally or also

intentionally from a malicious participant who is *disrupting* communication. This raises the central question addressed in this paper: How does each participant know when it is his or her turn to send? Note that many standard slot-reservation protocols as used, for example, in dynamic time-division multiple-access (TDMA) networks are not applicable, because they would compromise anonymity. The task of *slot-reservation* in DC-net is to agree on a sending schedule in a way that each participant knows when to send, but does not learn who is sending in the other slots.

Generally there are three different approaches to solve this problem. The first approach comprises so-called *reservation-map* methods. These methods employ a *scheduling vector* consisting of *slots*; each slot represents a future round with a corresponding index number. To reserve a round, a participant marks the corresponding slot in the scheduling vector as occupied. The second approach is to use *collision-resolution algorithms*. The third approach is to use secure multi-party computation to obtain a secret permutation that assigns rounds to participants.

Contributions of this paper. This paper presents a novel approach that belongs to the class of reservation-map methods. The general problem of this class of methods is that they essentially defer the problem of undetected collisions from the communication phase to a slot-reservation phase. One way to solve this problem is to use many more slots than participants, to keep the probability of collisions low. However, this leaves many communication slots unused and drastically reduces the throughput of the DC-net. We present *footprint scheduling* as a simple and efficient way to implement the slot-reservation phase without the loss of throughput. Footprint scheduling is the first scheduling algorithm to combine reasonable communication overhead that scales logarithmic in the number of participants, absence of computation overhead, and naturally handling participants joining and leaving the network.

Availability of results. To maximize re-usability of our results, we made the software used to produce simulation results publicly available at `https://github.com/25A0/DCnet-simulator`.

Notation. We write 0^B for the string that consists of B zeros. All logarithms in this paper are to the base 2.

Organization of this paper. Section 2 reviews the state of the art in scheduling for DC-net. Section 3 introduces footprint scheduling and Section 4 describes how to tune the parameters of footprint scheduling and compares its performance to previous approaches. Section 5 describes a protection mechanism against disruption. Section 6 summarizes the key strengths of footprint scheduling.

2 Existing scheduling methods

In this section we describe the previous approaches to scheduling in DC-net. As listed in the previous section, we group the algorithms into three categories: reservation maps, collision resolution, and secure multi-party computation (MPC).

Reservation maps. Reservation maps were already introduced as one possible way to handle scheduling in the original DC-net paper by Chaum [3]. The idea is to perform a separate scheduling phase to assign rounds to particular participants to avoid collisions. During this phase participants can reserve a round of DC-net by setting a bit of a *scheduling message* at the position corresponding to the round number. Note that also scheduling messages are sent through the anonymous DC-net channel; i.e., they are xored with all the shared keys of the other participants. Disruptions of the scheduling message can be detected with a certain probability by checking if the Hamming weight of the message is smaller or equal than the number of participants. The downside of this approach is that, because of the birthday paradox, the number of bits in the scheduling message must be quadratic in the number of participants to avoid collisions with high probability. Reservation maps are used by Herbivore, an implementation of DC-net presented by Goel, Robson, Polte, and Sirer in [9]. Herbivore optimizes the size of the scheduling message by allowing some collisions during message cycle depending on the message size (collisions for smaller messages are more likely). For performance comparison in Section 4 we also performed such optimization, however we decided for optimization that does not depend on the size of the message (See Appendix A for details).

The length of the scheduling message can be reduced if instead of bits representing rounds, one would use elements of the additive group of integers modulo m [12]. After all the scheduling messages of participants are added up, elements of value 0 indicate an unreserved round, elements of value 1 indicate a reserved round, all other values indicate collisions.

Collision resolution. The second approach proposed by Chaum in [3] is to use a contention algorithm with discrete time slots and resolve collisions by retransmitting the messages. A common example of such an algorithm is slotted ALOHA [14]. In Slotted ALOHA participants pick a time slot for transmission at will. Whenever a collision happens, participants wait for a random amount of time before they pick a new time slot for retransmission. The simplicity of the protocol is countered by the limitation of the transmission capacity of the network due to collisions.

One way to improve transmission capacity in collision resolution is through a technique called *superposed receiving*. The idea is to derive the retransmission schedule for collided messages from the result of a collision. New transmissions have to wait until a collision has been resolved. In [13, Sec. 3.1.2.3.2], A. Pfitzmann presents such an algorithm, which is an improvement of the tree algorithm that was independently proposed by Capetanakis in [2] and by Tsybakov and Mikhailov in [18]. Pfitzmann calls this algorithm *tree-like collision resolution with superposed receiving*; in the following, we refer to this algorithm as *Pfitzmann's algorithm*. Let the number of messages that collided be s, then the protocol guarantees that the collision will be resolved in exactly s retransmission steps. Additionally, this protocol guarantees fair usage of the channel for all sending participants. Note that this algorithm requires DC-net to be modified to work on integers modulo $m > 2$ instead of simple xors. This makes it possible to com-

pute the "average" of the collided messages (treated as an integer). Participants that sent a message smaller than the average retransmit; participants that sent a larger message wait. As soon as only two messages collided, only one participant retransmits; the other message is recomputed locally. A more detailed description of the algorithm can be found in [19] and [7, Sec 3.2.2].

Another algorithm, using a similar approach, was presented in [1] by Bos and Boer. It also computes a retransmission schedule for collided messages and requires s retransmissions after s messages collided. However, it has a larger overhead in header messages and is computationally more expensive than Pfitzmann's algorithm.

Note that these superposed-receiving techniques can also be applied to slot reservation as proposed by Waidner in [19]. Pfitzmann's algorithm is then used to resolve collisions of reservation messages. Each reservation message contains the number of the round in which a participant wants to send. With this approach the traffic load does not depend on the length of the messages transmitted through DC-net as in the original Pfitzmann's algorithm.

The first protective measure against disruption during the scheduling phase was presented by Waidner and B. Pfitzmann in [20,21]. The idea is to investigate the reservation phase in case of impossible results of the specific scheduling algorithm used. For example, in Pfitzmann's algorithm, the number of initially collided messages should be not more than the predefined maximum. To enable investigation, all the outputs during the scheduling phase are protected by output commitments. In the special phase called *palaver phase* any participant can start an investigation of the scheduling phase in order to detect disrupters.

Secure multi-party computation. In [10], Golle and Juels propose two new versions of DC-net that allow detection and identification of disrupters with high probability by using zero-knowledge proofs. They do not consider the problem of collisions (and thus reservation of rounds) in their solution; they comment that "the problem can be avoided through techniques like secure multi-party computation of a secretly distributed permutation of slots among players, but this is impractical".

Studholme and Blake propose in [15] a way to implement such a multi-party computation called *secret shuffle* by organizing a Mix-net with participants of DC-net serving as nodes. Encrypted round-reservation requests are transmitted through this Mix-net to obtain a secretly permuted vector of re-encrypted requests. Re-encryption is performed such that a participant can recognize his own request only after the permutation is completed. His reserved round number can be derived from the position of the request in the vector.

This idea is used in the Master's thesis of Franck [7], in which he derives a fully verifiable variant of DC-net. Later, verifiable DC-net was rediscovered in [6] and implemented under the name Verdict. The advantage of Verdict is that it allows switching between traditional DC-net and verifiable DC-net, depending on the presence of disruption. For scheduling, Verdict uses a similar approach as [15,7] and the same as in another implementation of DC-net by the same group, Dissent [5,22].

In [8], Franck proposes a scheduling for DC-net based on the collision-resolution protocol SICTA. This scheduling protocol is very similar to Pfitzmann's collision resolution algorithm, the only difference being that it operates with multiplication of ciphertexts instead of addition. The author proposes to use this algorithm to produce a secret shuffle of public keys of participants to establish a schedule. The protocol is non-deterministic; it achieves a maximum stable throughput (MST) of 0.924 messages per round. Disruption in the scheduling phase in this protocol is prevented by using zero-knowledge proofs.

3 Footprint scheduling

In this section, we introduce *footprint scheduling*. Footprint scheduling is similar to the map-reservation algorithm described by Chaum [3]; however, it requires significantly shorter reservation vectors and drastically decreases the likelihood of (undetected) collisions in these vectors.

In the map-reservation algorithm, the A active participants (i.e., participants who want to send a message in the next round) of a DC-net with a total of N users can reserve one out of S slots by inverting the corresponding bit in a reservation vector of S bits. The reservation vector is then transmitted through DC-net, and the resulting reservation vector, i.e., the xor of all the individual reservations, is broadcast to all participants. See also Section 2. An undetected collision occurs in this vector as soon as an odd number of participants attempts to reserve the same slot. This event is obviously undesirable, since it leads to collisions of messages during the sending phase. To decrease the probability of such an event, the original paper [3] suggests to choose the length of the reservation vector to be quadratic in the number of participants.

Footprints instead of bits. The first idea of footprint scheduling is to use $B > 1$ bits in the reservation vector to represent each slot of the schedule. The reservation vector is thus extended to a length of $B \cdot S$ bits. A participant attempts to reserve a specific slot by changing the corresponding B bits of the reservation vector to a random value $f \in \{0,1\}^B \setminus \{0\}^B$. This value is called his *footprint*. Figure 2a demonstrates an example of a reservation vector with 3-bit footprints. DC-net will broadcast the xor of all individual reservation vectors to the participants, just as for plain map reservation. If the reservation vector contains the footprint of a participant, it is likely that no other participant tried to reserve that slot. If instead the participant finds a different value, this indicates that at least one other participant tried to reserve the same slot. In Figure 2a one can see that participants `C`, `D` and `F` try to reserve the same slot. All three of them can recognize the collision since their original footprint s are not in the result of this round.

Scheduling cycles and message cycles. Using B bits per slot in the reservation vector alone would simply blow up the reservation vector by a factor of B. This is where footprint scheduling applies a second modification to reservation maps, which allows to drastically reduce the size of S (for example, to $S = 32$

A	000	000	000	000	000	000	110	000
B	000	000	000	101	000	000	000	000
C	000	011	000	000	000	000	000	000
D	000	100	000	000	000	000	000	000
E	000	000	000	000	001	000	000	000
F	000	010	000	000	000	000	000	000
R	000	101	000	101	001	000	110	000

(a) One scheduling round of footprint scheduling with 3-bit footprint s.

A	000	000	000	000	000	000	011	000
B	000	000	000	101	000	000	000	000
C	000	100	000	000	000	000	000	000
D	101	000	000	000	000	000	000	000
E	000	000	000	000	010	000	000	000
F	000	000	000	000	000	000	000	011
R	101	100	000	101	010	000	011	011

(b) The state of the reservation vector after the second scheduling round of footprint scheduling.

Fig. 2: The result of two scheduling rounds in footprint scheduling

for up to 10,000 participants). The idea is to iterate slot reservation through a *scheduling cycle* consisting of R *scheduling rounds*. In the first round, each participant just attempts to reserve a slot as described above. In each subsequent round, the behavior depends on whether the participant detected a collision in "his" slot in the previous round. If not, he will reserve the same slot again with a fresh random footprint. If the participant detected a collision, he flips a coin to choose between two possible actions. If the coin is 1, the participant backs off and does not continue to attempt to reserve any slot during this scheduling cycle. If the coin is 0, he tosses another coin[1]. If that second coin is 1, he stays in his slot and reserves it again with a fresh random footprint. Otherwise he randomly picks one of the slots that were left empty in the previous round (i.e., the ones that produced a zero xor of all footprints) and places a fresh random footprint in the corresponding slot. If no such empty slot exists, he backs off for the rest of the scheduling cycle.

In the last round of a cycle, all participants that detected a collision in their slot in the second but last round, back off and do not attempt to reserve a slot in the last round. This leaves the corresponding slots empty in the very last round. When the schedule is then executed, all empty slots can be skipped as in plain reservation maps.

Let us return to the example. After the first round, participants `A`, `B` and `E` have successfully reserved slots 7, 4 and 5, respectively. Participants `C`, `D` and `F` know that their reservation collided with reservation attempts by other participants. Slots 1, 3, 5 and 8 appear empty after the first round. Figure 2b demonstrates the reservation vector after the second round. Participants `D` and `F` have moved away from slot 2 to one of the empty slots, while participant `C` stayed in the first slot. Note that participants `A`, `B` and `E` placed a fresh footprint in the slots they successfully reserved during the first round. They will continue

[1] Note that tweaking the bias of these coin tosses is necessary to reach peak performance in large networks. The pseudocode description of the algorithm in Appendix B shows optimal probabilities for cases where users are allowed to reserve multiple slots.

to generate new footprints each round to reveal undetected collisions in case they occurred in the previous rounds.

By the end of the scheduling cycle several users hold reservations of slots in the following *message cycle*. The actual transfer of user messages in DC-net happens during this cycle. A message cycle has a maximum of S rounds, the maximum amount of slots users could reserve during the scheduling cycle.

Combining scheduling cycles and message cycles. The short length of a scheduling vector is advantageous since scheduling cycles and message cycles can now be combined to reduce latency in DC-net. For this, one has to have the number of scheduling rounds R be equal to (or smaller than) the number of slots S in the scheduling vector. Then the scheduling vector can be attached as a header to a message in the message cycle to agree on the schedule of the upcoming message cycle.

Multiple reservations. The activity rate of the network participants (i.e. the percentage of users who want to send data) will depend on the application for which a DC-net is used; for an anonymous file sharing application, the activity rate might hit 100% regularly, while a chat application might have a much lower activity rate on average. The algorithm that we described up to this point is not well-suited for networks with a very low activity rate. If there are less than S active participants, and each of them is allowed to reserve exactly one slot, then the remaining slots will be unused. This has two disadvantages: On the one hand, it limits the potential throughput of small, inactive networks. On the other hand, this leaks information about the number of actively sending participants in the network. If only 4 out of 16 slots are reserved at the end of a scheduling cycle, then it is very likely that there are exactly 4 actively sending participants.

Both disadvantages can be solved by allowing all participants to reserve up to S slots. Thus, at the beginning of a scheduling cycle, each participant picks up to S slots at random, and tries to individually reserve each of them, just as described above. It is important to note that different footprints have to be used for each slot.

A pseudocode description of footprint scheduling is given in Appendix B. It also shows the additional steps that have to be taken in order to allow participants to reserve multiple slots.

4 Benchmarks and comparison

This section shows how to optimize the configuration of footprint scheduling, and compares its performance to the performance of other scheduling algorithms. In the first part of this section, we will very briefly inspect the performance of footprint scheduling for different configurations in order to find optimal parameters. After that, we will compare its performance to the performance of Pfitzmann's scheduling algorithm and to Chaum's map-reservation scheduling algorithm.

Choice of parameters. There are three parameters that can be tweaked to minimize scheduling overhead: B, the number of bits per slot, S, the number

of slots, and R, the number of scheduling rounds per scheduling cycle. The scheduling overhead is measured in terms of the amount of scheduling data that each participant has to send for each successful reservation that the network achieves. During one scheduling cycle, each participant will send $B \cdot S \cdot R$ bits of scheduling data. At the end of the cycle, there will be $\hat{S}$ successful reservations. Ideally, all S slots were successfully reserved, so that $\hat{S} = S$. But $\hat{S}$ might also be smaller than S if there were undiscovered collisions or unused slots in the schedule. Thus, the overhead O can be measured by

$$O = \frac{S \cdot R \cdot B}{\hat{S}}. \tag{1}$$

Our optimization is mostly based on this formula, and we use simulations to test how different configurations perform. Schedule convergence is an important metric for this optimization. A messaging *slot* has converged if there is at most one participant who tries to reserve this slot. A *schedule* has converged if all S slots have converged. If a slot did not converge by the end of a scheduling cycle, then no participant can successfully send a message in that slot once the schedule is executed. It is thus crucial for the throughput of the algorithm that most of the slots converge.

We will start by minimizing $R \cdot B$. Figure 3 shows how many scheduling rounds are necessary to reach schedule convergence[2] for different values of B. Increasing B leads to a decrease in R, but it is easy to see that $B \cdot R$ is minimal for $B = 2$, regardless of the choice of S. Changing the network size does not affect this outcome.

In the previous simulation, the number of participants was fixed. But in order to determine an optimal value for R in general, we will now look at networks of various sizes. Figure 4 shows the number of scheduling rounds that are necessary to resolve all collisions for $B = 2$ bits and various values of S. The number of required rounds is clearly influenced by both, the network size and the number of slots. Although schedules converge faster on average when the number of slots is small, it is not recommended to choose $S < R$: Similar to Pfitzmann's scheduling algorithm, each message in footprint scheduling depends on the content of the previous message. Therefore, the scheduling data needs to be sent in individual packages. For $B = 2, S = 16$, these packages will only hold 32 bits of data. When this data is sent over the Internet, then the TCP-IP header of at least 32 Byte will add a massive overhead. But if $S \geq R$, then the current message cycle and the scheduling cycle for the following schedule can be interleaved as described in Section 3. This will decrease the *relative* overhead of the TCP-IP header and network delay. For this reason, we choose $S = 32$, which requires far less than 32 scheduling rounds, so that scheduling and message data can be interleaved.

Next, we will inspect the relation between the network size and the number of scheduling rounds. The dashed blue line in Figure 4 shows $log(N)$, with N being the number of participants in the network. For $S = 32$, no more than

[2] Note that, while it is easy to detect in a simulation whether all collisions have been resolved, it is not trivial to detect this in practice.

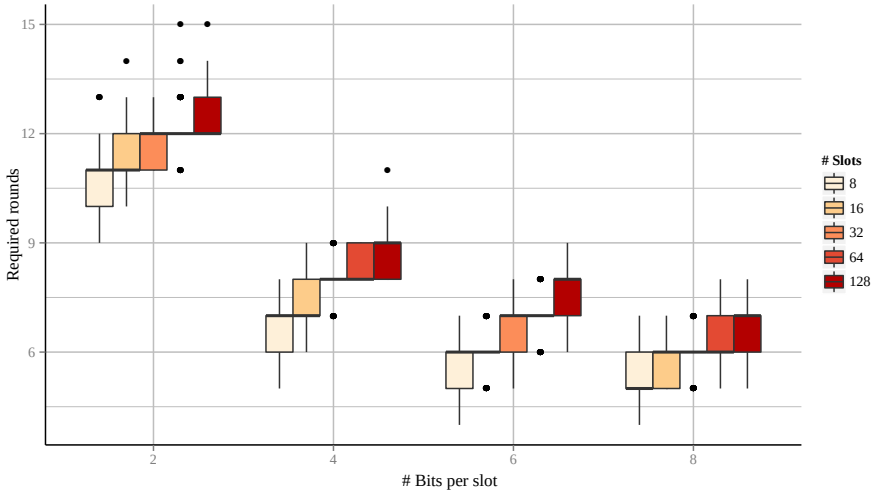

Fig. 3: The number of rounds that are necessary to resolve all collisions, for 5000 participants and different values of B and S.

$log(N)$ rounds are necessary on average for the schedule to converge. Thus, rather than having one fixed value for R, we can dynamically determine R based on the current size of the network.

Performance comparison. With this configuration, $S = 32, R = log(N)$, $B = 2$, we will now show how footprint scheduling performs compared to other scheduling algorithms. For this, we will show benchmark results for three scenarios: One with a very high activity rate, as it may occur for torrent downloads and video streaming, one with a very low activity rate, which could simulate instant-messaging, as well as two scenarios with intermediate activity rates.

We will compare the performance of footprint scheduling to the performance of Chaum's reservation map, as well as to the performance of Pfitzmann's collision resolution algorithm. We optimized the ratio between the number of participants and the number of slots for both algorithms using a series of simulations. We should note that the scheduling overhead of Chaum's reservation map depends heavily on the presence or absence of an estimation of the current activity rate. An abstract description of an algorithm that can be used to predict the activity rate is given in [9, pp. 10-11]. But especially in large networks where message cycles can take multiple minutes, the number of users might change drastically between two subsequent rounds. This makes it particularly difficult to predict the activity rate of the following cycle correctly. In our simulation, we measure the performance of Chaum's algorithm for the case that there is no estimation of the activity rate. Note that footprint scheduling is only configured based on the size of the network; it does not require an estimate of the num-

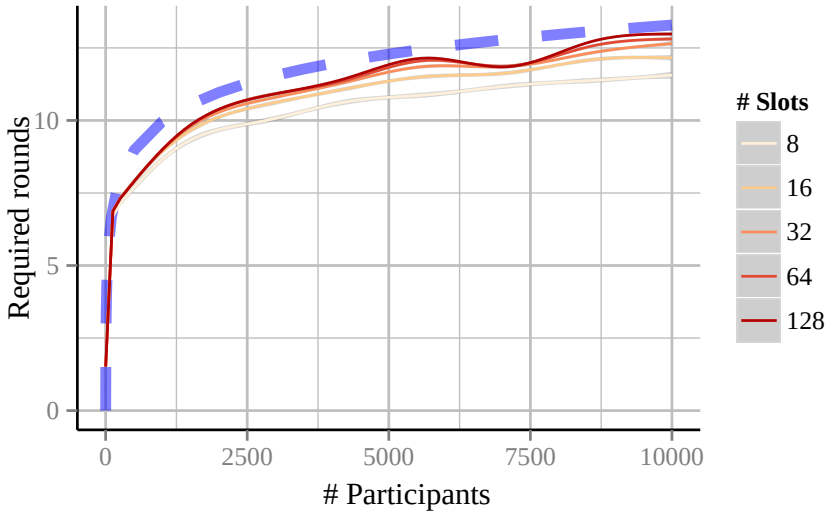

Fig. 4: The number of rounds that are necessary to resolve all collisions for various numbers of slots and networks of different sizes. These results were produced with 2 bit footprints.

ber of active users. More details on the optimization process can be found in Appendix A.

Performance is measured in terms of *scheduling overhead*, as defined in the beginning of this section. Note that the size of the message itself is not taken into account in our simulations because the absolute overhead to reserve a slot is not affected by the message size. Also, while Pfitzmann's algorithm could be used to resolve collisions between actual messages, we implement it for the scheduling purposes, which is in general more efficient for reasonable message sizes.

Before presenting results of simulation, we demonstrate in Table 1 theoretical estimations of scheduling overhead according to the formulas given in Table 2 in Appendix C. These formulas are valid in case of no collisions, which means that after completion of the scheduling protocol all slots are reserved successfully. In such conditions footprint scheduling is advantageous in all considered network sizes. However, results of simulations show somewhat different picture, which we discuss below.

In our simulations we do not consider the number of participants in the network to be larger than $10,000$. Significantly larger networks are almost always impractical because the amount of message data that is produced by the entire network grows quadratically with the number of participants.

Figure 5 shows the performance of the three scheduling algorithms in networks with different activity rates. The scheduling overhead of both, Pfitzmann's algorithm and footprint scheduling, scale with the activity rate of the network.

The overhead of Chaum's reservation maps increases with a decrease in the network's activity rate, since the available slots cannot be used as efficiently. Chaum's algorithm does, however, offer the lowest scheduling overhead for large networks. In theory, Pfitzmann's algorithm offers a scheduling overhead in large, active networks that is similar to what Chaum's reservation maps achieves, but we will address in Section 6 why Pfitzmann's algorithm is not practical in large networks.

In a network where the activity rate and the network size do not change drastically, the scheduling overhead can be minimized by choosing either Chaum's reservation maps or Pfitzmann's algorithm, depending on the exact characteristics of the network.

However, in a dynamic network where network size and activity rate change over time, footprint scheduling gives a better overall performance: While the activity rate is low, the network can benefit from the reduced scheduling overhead that footprint scheduling offers. In large networks and in networks with a very high activity rate, footprint scheduling still adds a slight increase in the scheduling overhead, compared to Chaum's reservation maps.

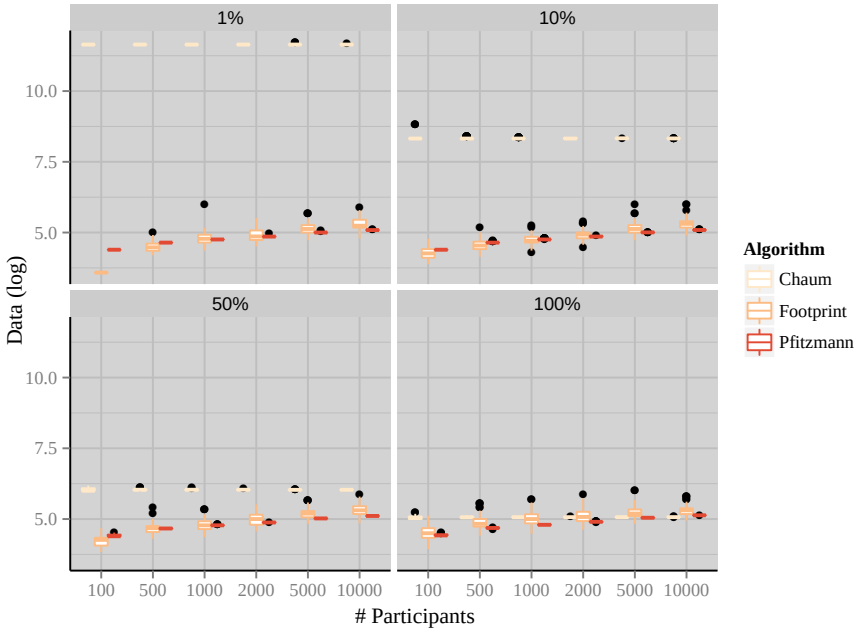

Fig. 5: The overhead of all three scheduling algorithm in networks with different activity rates.

	Participants						
Algorithm	100	200	500	1000	2000	5000	10,000
Footprint ($B = 2$)	13.29	15.29	17.93	19.93	21.93	24.58	26.58
Pfitzmann	24.93	27.93	31.9	34.9	37.9	41.86	44.86
Herbivore/Chaum ($A = 1\%$)	3200	3200	3200	3200	3200	3200	3200
Herbivore/Chaum ($A = 10\%$)	320	320	320	320	320	320	320
Herbivore/Chaum ($A = 50\%$)	64	64	64	64	64	64	64
Herbivore/Chaum ($A = 100\%$)	32	32	32	32	32	32	32

Table 1: Average scheduling overhead in Bytes (Based on the formulas given in Table 2).

5 Disruptions and footprint scheduling

Recall that disruptions are collisions that are intentionally induced by a denial-of-service attacker or a participant who attempts to increase his transmission bandwidth on the cost of the bandwidth of other participants. In this section we briefly describe a possible protection against an attacker who attempts to disrupt the scheduling phase of footprint scheduling.

The literature describes two approaches to cope with disruption in DC-net. The first approach is to open up special meaningless (*trap*) messages of participants after (suspected) disruption [3,1,21] and thus reveal which participants did not behave according to the protocol. The second approach is to use zero-knowledge proofs [10,5,6,8]. Our technique follows the first approach since it does not require any of the two computationally-secure variants of DC-net introduced in [10]. Additionally, scheduling messages can be opened without compromising anonymity of participants. This holds if the opened schedule is afterwards discarded and if the sending rates are constant, so sending wishes of a particular participant cannot be learned. One of the ways to achieve constant sending rates is to let users regularly reserve slots for dummy messages. Dummy messages in turn can be used as traps to protect the message phase from disruption.

The idea is to use a PRNG with a secret seed for all randomness that is required for footprint scheduling (i.e., slot positions, footprints and random choice to stay in a slot or back off). To prevent cheating, users are obliged to commit to the seed. Note that this is an obvious choice also for efficiency reasons. To protect against disrupters, each participant uses a new random seed for every scheduling cycle and commits to this seed before scheduling.

Whenever the decision is made to open a scheduling cycle, each participant publishes the seed used for this cycle. These seeds are checked against the com-

mitments and then the scheduling vectors of each round are recomputed and compared with the scheduling vectors that were previously obtained from the DC-net output. Note, keys and messages output by each individual participant are not opened and verified at this stage. If these recomputed scheduling vectors match, all participants followed the rules. If not, at least one of the participants did not follow the protocol. In order to find the disrupter, all participants reveal their keys used in the scheduling phase. To prevent disrupters from wrongly accusing honest participants by revealing an incorrect key, one can enforce that participants also commit to those shared keys in advance.

Such a technique provides performance improvements when the scheduling algorithm has a certain chance of undetected collisions (Footprint, Herbivore) for the following reason. Undetected collisions during the scheduling cycle lead to collisions during the message cycle. How can one efficiently distinguish an honest collision of messages due to such a problem and a disruption? Traps (non-meaningful messages that can be opened with no harm to anonymity) will not be helpful to answer this question for every single case of messages colliding. Opening of keys and rounds is too costly for such a check, ideally one would want to apply heavy methods only if it is known that there is a disrupter. Our method allows to perform a quick and efficient check if there was a disruption or not by opening only seeds used for generating footprints, and only after that decide if to open the scheduling cycle, which involves opening keys shared between users and verifying if individual outputs were made correctly.

6 Advantages of footprint scheduling

In this section, we provide a detailed description of the main advantages of the footprint algorithm.

As mentioned earlier, footprint scheduling inherits from the reservation-map algorithm. In particular, footprint scheduling involves no computational overhead for participants. The algorithm improves on reservation maps by reducing the probability of undetected collisions in the reservation vector. In networks with very high activity rate the cost for this improvement can be a very slight increase in scheduling overhead, depending on how many undetected collisions the reservation-map algorithm accepts. If message collisions are prohibitive or if the network does not have a very high activity rate, footprint scheduling noticeably *reduces* the scheduling overhead compared to reservation maps. For details see Section 4.

Further, unlike superposed receiving and MPC-based scheduling protocols, footprint scheduling naturally handles events of participants joining or leaving the DC-net during the schedule negotiation. When a participant disconnects, his reservation slot will appear free in the next round. Any participant that is in the process of resolving a collision can now move to this slot. Thus, footprint scheduling re-allocates slots that become available, even in the middle of a schedule cycle. At the same time footprint keeps scheduling and message cycles short,

which permits fast joining to the network. That in turn improves anonymity, allowing potentially a bigger anonymity set in the new cycle.

When using Chaum's reservation map, a good estimate of the network's activity rate is necessary in order to optimize the scheduling overhead (for details, see Section 4). With footprint scheduling, the success chance of a reservation attempt automatically increases if fewer participants bid for a slot in the next message cycle. Senders will be able to reserve a slot in fewer attempts if the activity of other participants goes down, and it will take more attempts if other participants become more active. This makes it unnecessary to estimate the activity rate.

Just like Pfitzmann's scheduling algorithm, footprint scheduling is an interactive protocol, in the sense that each message in the protocol depends on the content of the previous one. For Pfitzmann's algorithm, the protocol is completed after A successive messages, where A is the number of active network participants. In the case of footprint scheduling, the protocol is completed after S successive messages, where $S \ll A$ for large networks, as we showed in Section 4. In practice, network latency alone can be a major obstacle to complete a protocol of A messages in a large network. The following example shows a best-case scenario for a network with $10,000$ active participants: Assume that all participants live in major US cities. In this case, the average latency between them will be about $33ms$ on average at the time of this writing[3]. Thus, there will be a delay of at least $66ms$ between each message. With $10,000$ active participants, this means that the protocol is completed after $10,000 \cdot 66ms = 660s = 11min$.

Recall that Pfitzmann's scheduling protocol cannot be completed if a participant leaves before the protocol is completed. It is impractical to demand that not a single participant must leave the network over a period of 11 minutes, especially when some participants are connected via personal mobile devices like a phone or a laptop. Footprint scheduling does not suffer from this problem since each protocol run is completed much faster, but also since the protocol can be completed even if users disconnect from the network.

Footprint scheduling is also advantageous due to the fact that it hides the number of actively sending users in the network from an eavesdropping adversary. Such information can serve as a marker of upcoming social events; for example, the Tor network showed largely increased activity just before the Arab Spring[4].

One of the advantages of DC-net over Mix-nets (and onion routing) is that it hides the number of actively sending users due to the fact that all the users have to contribute to the network in order to facilitate anonymous sending. Unfortunately, previous efficient scheduling protocols either allow to estimate the number of active users by counting the number of empty slots in the scheduling messages, or they require to know the number of active users to operate.

Footprint scheduling disguises the number of active users as long as each user is allowed to reserve multiple slots. Even very small networks, the number

[3] http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html
[4] http://www.monitor.upeace.org/innerpg.cfm?id_article=816

of free slots does not give away the number of active users. An internal observer can gain a rough estimate of the number of active users over a longer period, based on the number of collisions that they experience. However, for an external observer, it is impossible to determine reliably whether there is a collision in any of the slots, since footprints change with every round. Further, for an external observer it is impossible to estimate if there were collisions in any of the slots since footprints change from round to round even if there were no collisions. In footprint scheduling the number of slots as well as discussion rounds does not change with the number of active users. Altogether, this prevents estimation of actively sending participants if footprint scheduling is used.

Last but not least, footprint scheduling has an advantage over other map reservation protocols due to fast and efficient method of verifying if a collision in message cycle was caused by an undetected collision in scheduling cycle or by a disruption. For details, see Section 5.

References

1. Jurjen N. Bos and Bert den Boer. Detection of disrupters in the DC protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, pages 320–327. Springer-Verlag Berlin Heidelberg, 1989. 5, 13
2. John I. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Transactions on Information Theory*, IT-25(5):505–515, 1979. 4
3. David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988. http://www.cs.ucsb.edu/~ravenben/classes/595n-s07/papers/dcnet-jcrypt88.pdf. 2, 4, 6, 13
4. David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981. www.freehaven.net/anonbib/cache/chaum-mix.pdf. 2
5. 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. http://dedis.cs.yale.edu/dissent/papers/ccs10/dissent.pdf. 5, 13
6. Henry Corrigan-Gibbs, David I. Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In *Proceedings of the 22nd USENIX Conference on Security*, pages 147–162. USENIX Association, 2013. http://dedis.cs.yale.edu/dissent/papers/verdict.pdf. 5, 13
7. Christian Franck. New directions for dining cryptographers. Master's thesis, University of Luxembourg, 2008. http://secan-lab.uni.lu/images/stories/christian_franck/FRANCK_Christian_Master_Thesis.pdf. 5
8. Christian Franck. Dining cryptographers with 0.924 verifiable collision resolution, 2014. http://arxiv.org/abs/1402.1732. 6, 13
9. Sharad Goel, Mark Robson, Milo Polte, and Emin Gun Sirer. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. Technical Report 2003-1890, Cornell University, 2003. http://www.cs.cornell.edu/People/egs/papers/herbivore-tr.pdf. 4, 10

10. Philippe Golle and Ari Juels. Dining cryptographers revisited. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 456–473. Springer-Verlag Berlin Heidelberg, 2004. http://www.iacr.org/cryptodb/archive/2004/EUROCRYPT/2389/2389.pdf. 5, 13
11. Steven J. Murdoch and George Danezis. Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy*, pages 183–195. IEEE, 2005. https://www.cl.cam.ac.uk/~sjm217/papers/oakland05torta.pdf. 2
12. Andreas Pfitzmann. How to implement ISDNs without user observability - Some remarks. Technical report, Department of Computer Science, University of Karlsruhe, 1985. Internal report 14/85. 4
13. Andreas Pfitzmann. *Diensteintegrierende Kommunikationsnetze mit teilnehmerüberprüfbarem Datenschutz*. PhD thesis, Fakultät für Informatik, Universität Karlsruhe, 1990. http://dud.inf.tu-dresden.de/sirene/publ/Pfit_88_0.pdf, http://dud.inf.tu-dresden.de/sirene/publ/Pfit_88_1.pdf, http://dud.inf.tu-dresden.de/sirene/publ/Pfit_88_2.pdf, http://dud.inf.tu-dresden.de/sirene/publ/Pfit_88_3.pdf, http://dud.inf.tu-dresden.de/sirene/publ/Pfit_88_4.pdf, http://dud.inf.tu-dresden.de/sirene/publ/Pfit_88_5.pdf, http://dud.inf.tu-dresden.de/sirene/publ/Pfit_88_6.pdf. 4
14. Lawrence G. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 5(2):28–42, 1975. http://www.disco.ethz.ch/alumni/pascal/refs/rn_1975_roberts.pdf. 4
15. Chris Studholme and Ian Blake. Multiparty computation to generate secret permutations. IACR Cryptology ePrint Archive: Report 2007/353, 2007. http://eprint.iacr.org/2007/353. 5
16. Paul F. Syverson, David M. Goldschlag, , and Michael G. Reed. Anonymous connections and onion routing. In *1997 IEEE Symposium on Security and Privacy*, pages 44–54. IEEE, 1997. www.onion-router.net/Publications/SSP-1997.pdf. 2
17. Tor project: Anonymity online. https://www.torproject.org/(accessedMar17, 2015). 2
18. Boris S. Tsybakov and V. A. Mikhailov. Free synchronous packet access in a broadcast channel with feedback. *Problemy Peredachi Informatsii*, 14(4):32–59, 1978. http://www.mathnet.ru/php/getFT.phtml?jrnid=ppi&paperid=1558&what=fullt&option_lang=eng (in Russian). 4
19. Michael Waidner. Unconditional sender and recipient untraceability in spite of active attacks. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 302–319. Springer-Verlag Berlin Heidelberg, 1990. 5
20. Michael Waidner and Birgit Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability (abstract). In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, page 690. Springer-Verlag Berlin Heidelberg, 1990. see also full version [21]. 5, 17
21. Michael Waidner and Birgit Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. Technical report, Universität Karlsruhe, 1998. See also abstract [20], http://dm.ing.unibs.it/giuzzi/corsi/Support/papers-cryptography/WaPf1_89DiscoEngl.pdf. 5, 13, 17

22. David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, pages 179–192. USENIX Association, 2012. http://dedis.cs.yale.edu/dissent/papers/osdi12.pdf. 5

A Optimization of Pfitzmann's algorithm and Chaum's reservation map

When Chaum's or Pfitzmann's algorithm is used for scheduling, the number of available slots has to be chosen appropriate to the size of the network. This section briefly explains how we optimized the ratio between the number of participants and the number of slots.

Assuming that participants choose random slots in the schedule, the collision probability in both algorithms underlies the birthday paradox. However, collisions do not need to be avoided at all costs: Pfitzmann's scheduling algorithm is capable of detecting them, and Chaum's algorithm should tolerate some collisions in order to limit the scheduling overhead.

Therefore the number of slots does not need to be quadratic in the number of participants. In fact, our simulations showed that it is sufficient if there is a linear relationship between the number of slots and the number of participants. Let i be the ratio between the number of slots S and the number of participants P, so that $S = i \cdot P$. Figure 6 shows the scheduling overhead for Pfitzmann's algorithm with different values of i. Pfitzmann's algorithm achieves minimal overhead for $i = 32$ across various network sizes. Based on these results, the simulations shown in Section 4 were executed using $i = 32$.

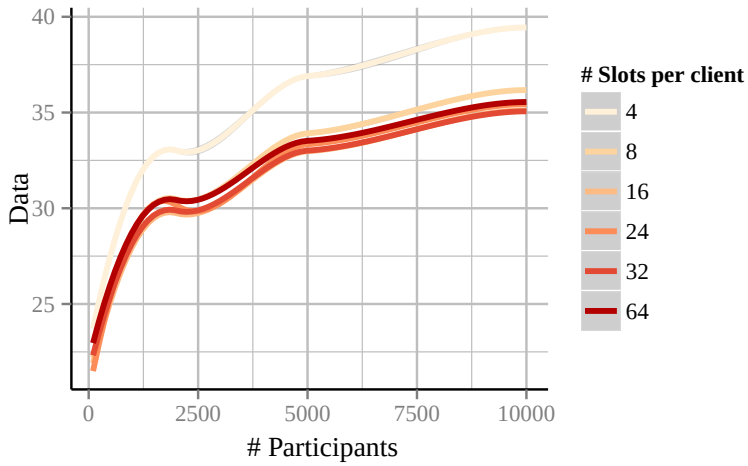

Fig. 6: The average scheduling overhead produced by Pfitzmann's algorithm for different numbers of slots per participant.

Chaum's algorithm cannot detect collisions and therefore needs to minimize the chance of collisions to occur. Figure 7 shows the fraction of participants that will experience a collision in their reservation attempt, given different values of i.

We aimed for a configuration where no more than one in 20 reservation attempts would fail, which requires about 32 slots per participant.

The results for Chaum's algorithm shown in Section 4 were achieved with $i = 32$. If a higher success rate is desired, then the number of slots must be increased more.

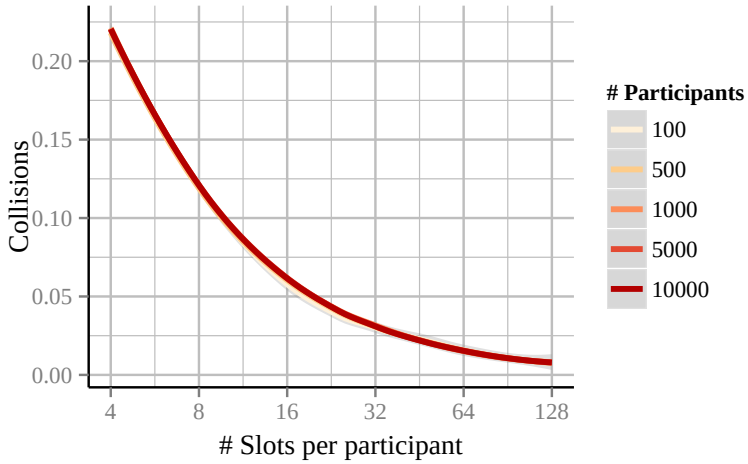

Fig. 7: The fraction of participants that will experience a collision in Chaum's reservation map algorithm, for different ratios between the network size and the number of slots.

B Pseudocode description of footprint scheduling

Algorithm 1 Footprint scheduling from the perspective of one participant A

Parameters: Number of footprint bits B, number of participants N, number of slots S per message cycle

Output: A vector $D \in \{0,1\}^S$, indicating for each slot whether it can be used for sending.

```
R ← log(N)
D ← {1}^S                          ▷ Vector indicating which slots can be used for sending
f ← {0,1}^B \ {0}^B                ▷ Set of possible footprints
F ←_R f^S                          ▷ Vector holding footprints for each slot
V ← SlotReserve(D, F)              ▷ First round of the scheduling cycle
for i from 1 to R − 2 do           ▷ Rounds 1 to R − 2
    for j from 0 to s − 1 do
        if D[j] = 0 then
            continue
        end if
        if V[j] ≠ F[j] then        ▷ Reservation attempt failed
            c1 ←_R [0, 1)          ▷ Biased coin toss
            if c1 < 0.7 then
                D[j] ← 0           ▷ Back off
            else
                c2 ←_R {0, 1}
                if c2 = 1 then     ▷ Try same slot again
                else               ▷ Empty slot available?
                    I ← {s′ | D[s′] = 0 and V[s′] = 0}
                    D[j] ← 0
                    if I ≠ ∅ then  ▷ Pick empty slot
                        s′ ←_R I
                        D[s′] ← 1
                    end if
                end if
            end if
        end if
    end for                        ▷ Generate new footprints
    F ←_R f^S
    V ← SlotReserve(D, F)
end for
for j from 0 to s − 1 do           ▷ Last round
    if V[j] ≠ F[j] then
        D[j] ← 0
    end if
end for
F ←_R f^S
SlotReserve(D, F)
return D
```

Algorithm 2 Procedure for a slot-reservation attempt in footprint scheduling

```
procedure SlotReserve(D, F)
    V_A ← {{0}^B}^s
    for i from 0 to s − 1 do
        if D[i] = 1 then V_A[i] ← F[i]
        end if
    end for
    Broadcast V_A through DC-net
    Receive V (xor of all individual scheduling vectors) from DC-net
    return V
end procedure
```

C Overview of main scheduling methods

Table 2 compares 3 main scheduling methods discussed in this paper with footprint scheduling. Some explanations of the compared properties:

- *Guaranteed sending* - indicates if a user is guaranteed to have a slot reserved after one run of scheduling protocol with no adversaries present.
- *Equal throughput* - indicates if all users are assigned the same amount of reserved slots or not. Note, that Pfitzmann's algorithm used for collision resolution provides equal throughput for all sending users. However, as a reservation method it does not due to possibility of several users choosing the same slot number to reserve.
- *Adopts to load change* - indicates if the protocol can adopt to changing number of active users between two protocol runs to avoid too many empty slots or too many unsuccessful reservation attempts.
- *Disconnected users* - shows if the protocol can handle users disconnecting during run of the scheduling protocol. For example, Pfitzmann's algorithm would have to completely restart after a user is dropped out.
- *Collisions* - describes how collisions during run of the scheduling protocol are handled.
- *Scheduling overhead* - demonstrates how much data a single user has to send in order to reserve a single slot. To simplify estimations, given formulas are calculated under assumption that there are no collisions and no adversaries present. In footprint scheduling a user sends BRS amount of data (over R rounds), that leads to S slot reservation in the ideal case. Thus, for a single slot reservation, a user has to send BR data. Note, that it differs from the formula 1 in Section 4 in the assumption that reservation protocol leads to S successful reservations. According to our performance estimations, one can use R equal to $log(N)$. In Chaum's algorithm a user sends once N^2 amount data, leading to A slot reservations in ideal case. Thus, the single slot reservation requires $\approx \frac{N^2}{A}$. After optimization described in Appendix A the formula becomes $\approx \frac{32N}{A}$. Note, optimization comes at the cost of tolerating collisions during message cycle. In Pfitzmann's algorithm each can choose

a slot with a number up to N, however one has to choose a larger modulo $m = N^2$ to avoid undetected collisions when adding up choices of all users. Thus one needs $log(N^2)$ bits to represent each slot number. To avoid choosing same slot number, one has to multiply N^2 with 32 (See Appendix A). In addition to that, each schedule message is appended with $log(N)$ bits for counting the number of collided messages. For Dissent protocol we provide simplified formula that counts only large messages sent through the network.

– *Hides number of active users* - indicates if the protocol allows an external observer to estimate from the scheduling cycle how many users are going to send in the DC-net.

Property	Footprint	Chaum/Herbivore	Pfitzmann	Dissent
Guaranteed sending	-	-	-	+
Equal throughput	-	-	-	Guaranteed
Adopts to load change	Inherent	Estimated	Inherent	Manual configuration by group leader
Disconnected users	Compensated	Unaffected/Increases number of empty slots	Protocol fails	Protocol fails
Collisions	Resolved with high probability	Prevent reservation	Prevent reservation	Impossible
Required operations	XOR	XOR	Addition modulo $m > 2$	Public key encryption, signing, zero-knowledge proof
Scheduling overhead [5]	$B \times log(N)$, See[6]	$\approx \frac{32N}{A}$, See[7]	$log(32N^2) + log(N)$, See[8]	$\approx \frac{(A-1)(\tilde{J}+2A\tilde{J})}{A^2}$, See[9]
Amount of rounds	Adjustable, $R = log(N)$	One	A	Not applicable
Achieves privacy	Unconditional	Unconditional	Unconditional	Computational
Hides number of active users	+	-	-	-

Table 2: Overview of main scheduling methods.[10]

[5] For numerical examples see Table 1

[6] Recommended $B = 2$, thus the formula is $2R$

[7] Optimized, See Appendix A

[8] Optimized, See Appendix A

[9] Simplified estimation (very optimistic)

[10] **Notation**: N - number of users; A - number of active users during the scheduling cycle
R - number of scheduling rounds per scheduling cycle in footprint scheduling
$\tilde{J}$ - amount of bits the ciphertext of length J increases after public-key encryptions