Incentivizing Collaboration in Privacy-preserving Mechanisms Using Scrip System - Simulation

Cihangir Tezcan
Tutors: Mathias Humbert and Dr. Mohammad Hossein Manshaei
Instructor: Prof. Jean-Pierre Hubaux

École Polytechnique Fédérale de Lausanne
January 11, 2011

1 Introduction

Novel mobile systems, such as location-based services, mobile social networks or participatory sensing, leverage on increasing capabilities in computation, communication, storage and sensing of mobile devices, as well as location awareness (GPS), in order to provide mobile users with services or other kinds of rewards. In order to benefit from these, mobile users are required to share location information and sensitive data with a central server or other peers. This leads to many issues, among which privacy is one of the most critical. In [1], the authors propose a new privacy-preserving system for data aggregation in participatory sensing, based on data slicing and mixing. However, this scheme induces a communication overhead (and thus more energy consumption) to users collaborating in the privacy-preserving mechanism. As battery is a critical resource in all mobile devices, self-interested users may not be willing to cooperate and even free-ride (i.e., benefit from the system without paying its costs). In order to prevent free-riding in P2P networks, some researchers have proposed to make use of virtual currency, called scrip [2, 3].

In this mini-project, we would like to adapt scrip system to the privacy-preserving scheme proposed in [1] for participatory sensing. The amount of money (scrip) deployed in the system has been shown to have great impact on the efficiency of P2P networks. The distribution of money among the nodes is also a crucial point that must be taken into account. In order to solve these problems, we would like to implement a simulation infrastructure and observe the changes in the system by running simulations. In this study, we used the C programming language to implement our simulation infrastructure and used the command-line program gnuplot [4] to plot graphs of the simulation data.

This report is organized as follows: In Section 2, we define the variables and functions of the simulation, discuss parameter selection and plotting simulation data with gnuplot. Simulation results are provided in Section 3. In Section 4, we conclude our report with a summary and a future work. Note that the C codes of our simulation are provided in the Appendices.

2 Simulation Infrastructure

2.1 Our Model

We define our scrip system with *nnodes* many nodes which is similar to the scrip system defined in [3]. Main difference between these systems is that, in [3] a service is provided to a node by another node. However, in our model, nodes slice each piece of their data into $slices + 1$ smaller pieces of equal size. They keep one piece to themselves and to preserve their privacy, they send remaining *slices* many pieces to their neighbours, which are called *cover nodes* (We assume that the cover nodes aggregate all received slices and send the result to the aggregation server). Hence, the system of [3] can be seen as 1-1 and our system can be seen as 1-n.

We assume that nodes have *type* t and define a type by a tuple t=(α_t,β_t,γ_t,δ_t,ρ_t) as in [3] where

- α_t is the cost of satisfying a request,
- β_t is the probability that a node can satisfy the request of another node,
- γ_t is the utility a node gains for having a request satisfied,
- δ_t is the rate at which a node discounts utility,
- ρ_t is the relative request rate.

When a service is performed by *slices* many cover nodes for a requester node, cover nodes lose α utility depending on their type and each of them gain 1 scrip. On the other hand, requester node gains γ utility and loses *slices* many scrips.

Our simulation runs until *n_transactions* many successful transactions occur. In each round, a node is selected depending on ρ values of the nodes. If the scrip amount of the selected node is less than *slices*, it cannot request a service. Otherwise, it randomly selects a neighbour and asks for transmission. If it can find *slices* many cover nodes that are willing to perform the service, then the data transmission occurs and the requester node sends 1 scrip to every cover node.

In [3], total expected *social welfare* for a homogeneous population (i.e. each node is of type t for a fixed t) is defined as $(1 - M_0)(\gamma_t - \alpha_t)/(1 - \delta_t)$

where M_0 is the fraction of nodes that has 0 scrip. We define the total expected social welfare in our system in a similar way. Let γ_{avg}, α_{avg}, δ_{avg} be the weighted average of the γ, α, δ values of all types and let M_{brokes} represent the fraction of nodes whose scrip amount is less than *slices*. When a request is satisfied, the social welfare increases by $\gamma_{avg} - slices \cdot \alpha_{avg}$ in average. If we assume that there are always enough cover nodes to perform the service, in each round the expected increase in the social welfare is $(1 - M_{brokes})(\gamma_{avg} - slices \cdot \alpha_{avg})$. Thus, the total expected social welfare summed over all rounds is $(1 - M_{brokes})(\gamma_{avg} - slices \cdot \alpha_{avg})/(1 - \delta_{avg})$ where δ_{avg} is the average discount factor.

Our simulation software consists of two files: 'scrip.c' and 'scrip.h'. Variables and main functions are defined in 'scrip.c' file and functions for printing results on the screen or to a file is defined in 'scrip.h' file. We used the C programming language for these codes and they are provided in the Appendices.

2.2 Parameters

In order to start the simulation, the user is asked to input parameters in the following order:

1. Number of nodes
2. Length of the x-coordinate
3. Length of the y-coordinate
4. Coverage radius of a node
5. Number of slices
6. Average amount of money
7. Number of transactions
8. Output frequency
9. Number of types
   (a) Ratio
   (b) Threshold
   (c) Alpha (Cost of satisfying a request)
   (d) Beta (Probability of satisfying a request)
   (e) Gamma (Utility the agent gains)
   (f) Delta (Rate at which an agent discounts utility)
   (g) Rho (Relative request rate)

Users should take into account following properties when entering parameters:

1. The number of nodes in the system is upper bounded by $10,000$ (However, this upper bound can be changed by editing 'scrip.c' file).

2. For the input values of the lengths of the coordinates x and y, nodes are randomly placed in the XY-plane where the x-coordinate of a node is in $[0, x)$ and y-coordinate is in $[0, y)$. Two nodes cannot be placed at the same position. Therefore, depending on the number of nodes in the simulation, the values x and y should be chosen big enough.

3. Nodes can communicate only with the nodes that are inside their coverage radius. Such nodes are called neighbours. Note that the current version of the simulation does not allow h-hop data transfer and the simulation does not start if there exists a node that has no neighbours. To allow every node to hear each other, the user can simply enter '0' as the coverage radius.

4. For the input average amount of money m, half of the nodes initially has $\lfloor \frac{m}{2} \rfloor$ scrips each and the remaining half has $m + \lfloor \frac{m}{2} \rfloor$ scrips each. This property is also true for each type of nodes. For instance, if there are 20 nodes of Type 1 and 80 nodes of Type 2 in the system with average money 8, then 10 nodes of Type 1 and 40 nodes of Type 2 have 4 initial scrips and the remaining nodes have 12 initial scrips.

5. If a node has less scrips than the number of slices, then it cannot transmit data. Hence, the choice of the number of slices should be made according to average amount of money. Note that if the number of slices is higher than $m + \lfloor \frac{m}{2} \rfloor$, no one can transmit data and system crashes.

6. Simulation runs until the number of successful data transmissions equals to the user's input value of number of transactions. We are interested in the fraction of nodes having i scrips at the end of the simulation and after some point, we expect to see these values to converge to a limit value. Hence, number of transactions should be big enough.

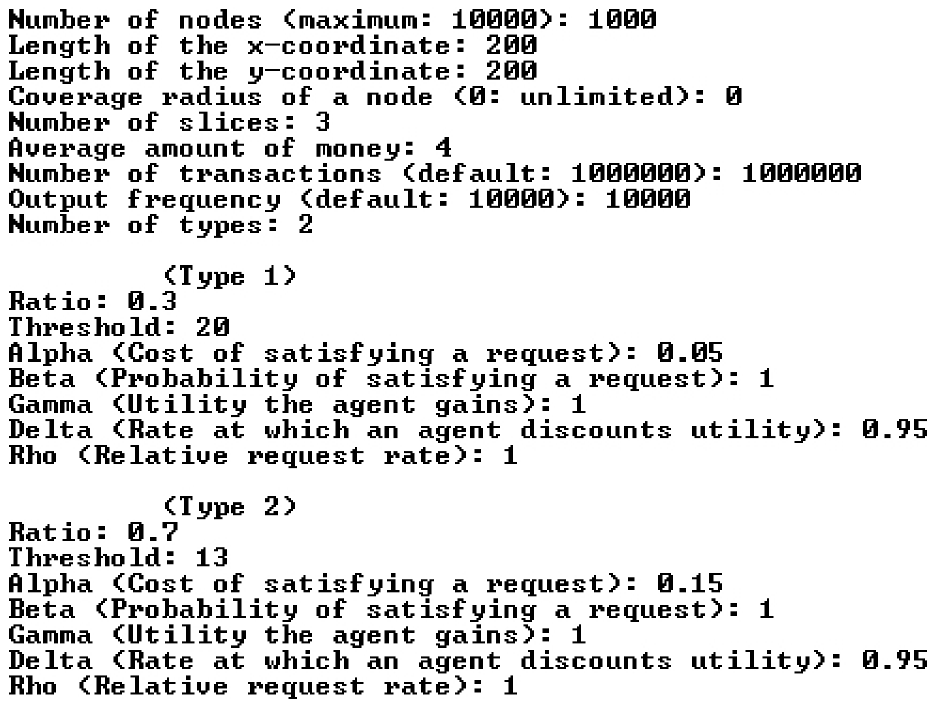
7. The scrip data of the simulation are recorded to 'scrips.txt' and 'social_welfare.txt' file once when $outputfrequency$ many transactions are done. Note that writing simulation data to the hard drive too frequently may increase the running time of the simulation dramatically.

8. Sum of the ratios of the types must be equal to 1.

9. Beta values must be real values in $[0, 1]$ since they are probabilities.

10. Rho values should be normalized to integer values.

An example of user input is provided in Figure 1.



```
Number of nodes (maximum: 10000): 1000
Length of the x-coordinate: 200
Length of the y-coordinate: 200
Coverage radius of a node (0: unlimited): 0
Number of slices: 3
Average amount of money: 4
Number of transactions (default: 1000000): 1000000
Output frequency (default: 10000): 10000
Number of types: 2

         (Type 1)
Ratio: 0.3
Threshold: 20
Alpha (Cost of satisfying a request): 0.05
Beta (Probability of satisfying a request): 1
Gamma (Utility the agent gains): 1
Delta (Rate at which an agent discounts utility): 0.95
Rho (Relative request rate): 1

         (Type 2)
Ratio: 0.7
Threshold: 13
Alpha (Cost of satisfying a request): 0.15
Beta (Probability of satisfying a request): 1
Gamma (Utility the agent gains): 1
Delta (Rate at which an agent discounts utility): 0.95
Rho (Relative request rate): 1
```

Fig. 1: Example inputs

2.3 Variables

Integer Valued Variables: The following integer valued global variables are used in the simulation:

x: Length of the x-coordinate where nodes can be placed. Hence, X-coordinate of a node is in the range $[0, x)$.

y: Length of the y-coordinate where nodes can be placed. Hence, Y-coordinate of a node is in the range $[0, y)$.

distance: The coverage radius of a node.

number_nodes: Maximum number of nodes that can be used in the simulation. Default value is $10,000$. This upper bound can be changed only by updating the 'scrip.c' file. Note that there can be at most $x * y$ many nodes in the simulation.

max_types: Maximum number of node types that can be used in the simulation. Default value is 100. This upper bound can be changed only by updating the 'scrip.c' file.

max_slices: Maximum number of slices that a data package contains. Default value is 100. This upper bound can be changed only by updating the 'scrip.c' file.

nodes()(): The coordinates of the nodes in the XY-plane are stored in this array. For instance, the coordinates of the node $N5$ are $(X, Y) = (nodes(5)(0), nodes(5)(1))$.

neighbours()(): Neighbours of every node are stored in this array. For instance, the 2nd neighbour of $N6$ is $neighbours(6)(1)$.

n_neighbours(): Number of neighbours of a node. For example, $N6$ has $n\_neighbours(6)$ neighbours.

scrip(): Number of scrips of a node. For example, $N6$ has $scrip(6)$ many scrips.

n_transactions: Number of successful data transmission.

history(): Keeps a track of the nodes to which a slice of data is sent in order to avoid sending more than 1 slice to the same cover node.

available_nodes: Number of neighbours that are not contacted yet.

minimum_n_neighbours: The minimum value of the number of neighbours (i.e. $\min n\_neighbours(i)$ where $i \in [0, nnodes)$). If there exists a node that has no neighbours (i.e. $minimum\_n\_neighbours$=0), then the simulation does not start.

total_rejections: Number of rejected transmissions

n_brokes: Number of nodes whose scrip is less than the value of $slices$. In other words, this is the number of nodes that does not have enough scrip to transmit data.

outputs: This counter is increased when the scrip data is recorded to a file. Hence, it shows the amount of data that is stored.

slices: Number of slices a package contains.

acknowledged(): Cover nodes that agree to transmit a data slice is kept in this array. For example, 0th slice will be transmitted to $acknowledged(0)$.

randomization(): In each round, a node is selected with probability proportional to rho. This array keeps the list of nodes for random selection.

modulo: When determining the node to send data, random numbers are generated modulo this number. (i.e. rand()%modulo) See *prepare_table_random_selection* function for more information.

n_rich: Number of nodes that have scrips higher than or equal to their thresholds.

tthreshold(): Represents the threshold of each type.

nnodes: Number of nodes.

m: Average amount of money.

types: Number of types.

transaction: Number of transactions required to complete the simulation.

output_frequency: Frequency of recording the scrip data to the output file. Hence, scrip data is recorded $\lfloor \frac{transaction}{output\_frequency} \rfloor$ many times.

ntype(): Type of each node.

Real Valued Variables: The following real valued global variables are used in the simulation:

ratio(): Ratio of each type.

alpha(): Cost of satisfying a slice transmission request of each type.

beta(): Probability that a node acknowledges a slice transmission request. This value is defined for each type.

gamma(): Amount of utility that a node gains for having a request satisfied. This value is defined for each type.

delta(): Utility discount rate of a node. This value is defined for each type.

rho(): Relative data submission rate. This value is defined for each type.

average_gamma: Weighted average of *gamma* values of each type. This value is used when calculating social welfare.

average_alpha: Weighted average of *alpha* values of each type. This value is used when calculating social welfare.

average_delta: Weighted average of *delta* values of each type. This value is used when calculating social welfare.

2.4 Functions

scrip.c File: Following functions are defined in 'scrip.c' file:

define_nodes: This function puts *nnodes* many nodes on the XY-plane. The coordinates of the nodes are randomly generated.

prepare_variables: Initializes some variables.

detect_neighbours: Every node detects other nodes which are inside the coverage radius. The $neighbours()$ array keeps this list of detected nodes.

check_consistency: This function checks the user's input for errors. Simulation does not start if there are any error. It reports an error if

1. $beta(i) \notin [0, 1]$ for some $i \in [0, types)$,

2. $ratio(i) \notin [0, 1]$ for some $i \in [0, types)$,
3. $\sum_{i=0}^{types} ratio[i] \neq 1$,
4. $nnodes > x * y$.

distribute_money: This function distributes initial scrips to nodes depending on the average scrip amount m. Note that half of the nodes has $\lfloor \frac{m}{2} \rfloor$ scrips and the remaining half has $m + \lfloor \frac{m}{2} \rfloor$ scrips.

volunteers: Returns the number of nodes that are not contacted yet for slice transmission acknowledgment.

send_data: This function is called when a data package is sent. Hence, it increases the scrip amount of the *slices* many cover nodes which acknowledged the transmission. Moreover, the requester node loses *slices* many scrips. This function is called inside the *exchange_data* function if the requester finds *slices* many cover nodes that acknowledges data transmission.

acknowledge_transmission: With this function the requester node tries to find *slices* many cover nodes that acknowledges data transmission.

exchange_data: First, it selects a node to request data transmission randomly (depending on the data submission rate *rho* of the nodes). Then checks if there are *slices* many cover nodes that acknowledges data transmission, using *acknowledge_transmission* function. If so, the data transmission and exchange of scrips is performed by calling the *send_data* function.

prepare_table_random_selection: In each round, a node is selected to request data transmission and the selection depends on the data submission rate of the nodes. This function creates a list of the nodes $randomization()$ for this selection. For example, if node N_0 has $rho = 1$, then it appears only once in this list and if N_1 has $rho = 6$, it appears 6 times in the list. Hence, the probability of selection of N_1 is 6 times the probability of selection of N_0.

calculate_brokes: Calculates the number of nodes whose scrip is less than *slices* and records this number to *n_brokes* variable. It also calculates the number of nodes whose scrip equals to its *tthreshold* and records this number to *n_rich* variable.

input_parameters: Asks the user to input parameters of the simulation. The parameters are asked in the following order: *nnodes*, *x*, *y*, *distance*, *slices*, *m*, *transaction*, *output_frequency*, *types*.

And for each type, following parameters are asked: *ratio*, *tthreshold*, *alpha*, *beta*, *gamma*, *delta*, *rho*.

main: This is the main function that runs the simulation. First, it seeds the pseudo random number generating function with the current time in order to have different random numbers in each simulation. Second, it asks the user to input parameters using *input_parameters* function. The validity of the input parameters are checked with the *check _consistency* function. Then, *prepare_variables* function is called to initialize some values and *define_nodes* function is called to generate nodes. Generated nodes recognize their neighbours with *detect_neighbours* function. The initial scrips are distributed with the *distribute_money* function and it is recorded to 'scrips.txt' file using *record_scrips* function. The list for randomly selecting nodes that want to transmit data is generated with the *prepare_table_random_selection*. For *transaction* many rounds, *exchange_data* function is called to perform the simulation.

scrip.h File Following functions are defined in 'scrip.h' file:

print_scrip: Prints the scrip amount of each node on the screen.

print_n_neighbours: Prints the number of neighbours of each node and the minimum number of neighbours on the screen.

print_neighbours: Prints the neighbours of each node on the screen.

print_nodes: Prints the positions of each node on the XY-plane on the screen.

fprint_nodes: Prints the positions of each node on the XY-plane on the file 'nodes.txt'. Moreover, it records gnuplot instructions for plotting the positions of the nodes and their coverage on the XY-plane to 'gnu_nodes.txt' *record_scrips:* Records the number of nodes that has i scrips for any $i \geq 0$ to the 'scrips.txt' file.

record_social_welfare: Records the social welfare to the 'social_welfare.txt' file where social welfare is calculated as

$$\frac{(1 - \frac{n\_brokes}{nnodes}) \cdot (average\_gamma - slices \cdot average\_alpha)}{(1 - average\_delta)}.$$

fprint_parameters: Records the user input parameters to 'parameters.txt' file.

2.5 Graphs with gnuplot

Scrip data of the simulation is recorded to the output file 'scrip.txt' for $\lfloor \frac{transaction}{output\_frequency} \rfloor$ many times and each time they are recorded as a different data set. We use gnuplot software to plot these graphs. First, we write the instructions for plotting the graphs to a file named 'gnu_scrips.txt'. For example, the following instructions can be used to plot first 5 data sets:

```
set term pdf size 3.00in, 3.00in
set pointsize 0.5
set xrange[0:55]
set yrange[0:1010]
set xlabel 'Number of scrips'
set ylabel 'Number of nodes'
set style line 1 lt 1 lw 5
set title 'Scrip Distribution at Time slot:0
set output 'scrips000.pdf'
plot 'scrips.txt' index 0:0 using 1:2 ti '' with imp ls 1
set yrange[0:200]
set title 'Scrip Distribution at Time slot:1
set output 'scrips001.pdf'
plot 'scrips.txt' index 1:1 using 1:2 ti '' with imp ls 1
set title 'Scrip Distribution at Time slot:2
set output 'scrips002.pdf'
plot 'scrips.txt' index 2:2 using 1:2 ti '' with imp ls 1
set title 'Scrip Distribution at Time slot:3
set output 'scrips003.pdf'
```

```
plot 'scrips.txt' index 3:3 using 1:2 ti '' with imp ls 1
set title 'Scrip Distribution at Time slot:4
set output 'scrips004.pdf'
plot 'scrips.txt' index 4:4 using 1:2 ti '' with imp ls 1
set title 'Scrip Distribution at Time slot:5
```

Once the 'scrip.txt' and the 'gnu_scrips.txt' are in the same folder with the 'gnuplot.exe' (which is generally <installation folder>/binary) we call the gnuplot with the following command:

```
gnuplot.exe "gnu_scrips.txt"
```

Note that these example instructions that we defined above plots the first 5 data sets to 5 different PDF files. The *index* command is used to specify which data set we want to use for plotting the graph. The *set xrange[0:55]* command defines the length of the x-coordinate in the graph and this value should be changed according to the thresholds of the types. Similarly, the *set yrange[0:200]* command defines the length of the y-coordinate in the graph and this value should be changed according to the total amount of scrips in the system.

3 Results

3.1 Scrip Distribution

We are interested in the distribution of money after the simulation is run long enough and we also would like to see the effect of the number of slices on this distribution. Figures 2, 3, 4, and 5 show the distribution of money after $1,000,000$ successful transactions where number of slices are chosen as 1, 2, 3, and 4, respectively. The remaining parameters are fixed for all of the four figures and they are chosen as follows:

```
Number of nodes: 1000
Length of the x-coordinate: 200
Length of the y-coordinate: 200
Coverage radius of a node: 0
Average amount of money: 10
Number of transactions: 1000000
Output frequency: 10000
Types: 2

(Type 1)
```

```
Ratio: 0.3
Threshold: 50
Alpha: 0.05
Beta: 1
Gamma: 1
Delta: 0.95
Rho:1

(Type 2)
Ratio: 0.7
Threshold: 50
Alpha: 0.15
Beta: 1
Gamma: 1
Delta: 0.95
Rho:1
```

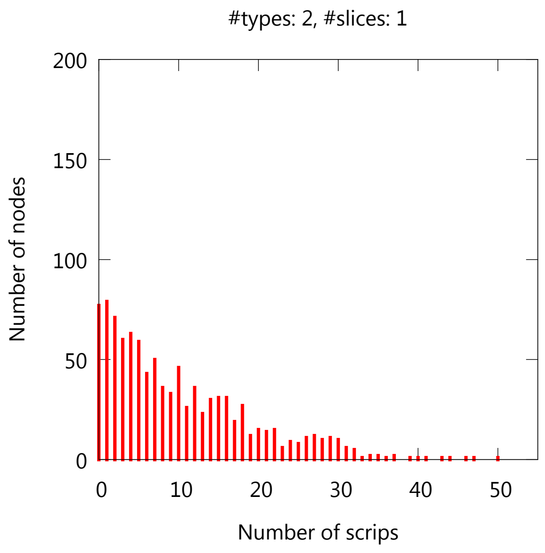


Fig. 2: Scrip distribution after 1,000,000 transactions

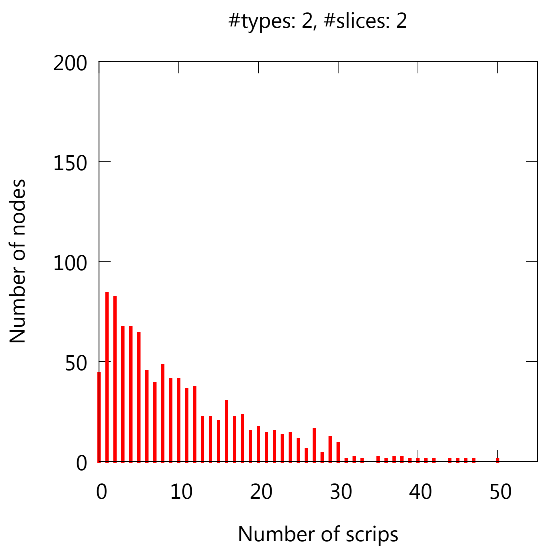


Fig. 3: Scrip distribution after 1,000,000 transactions

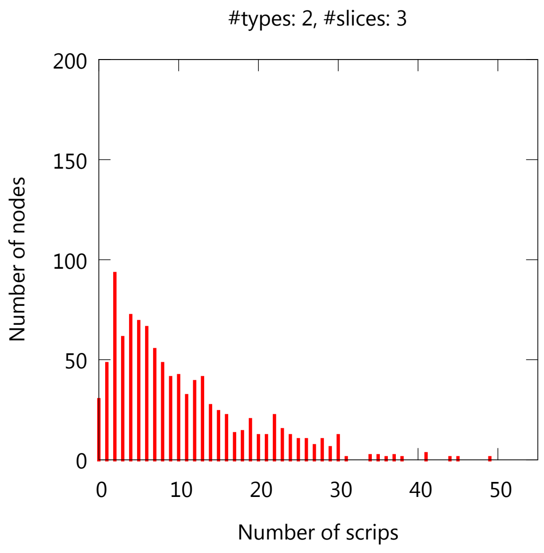


Fig. 4: Scrip distribution after 1,000,000 transactions

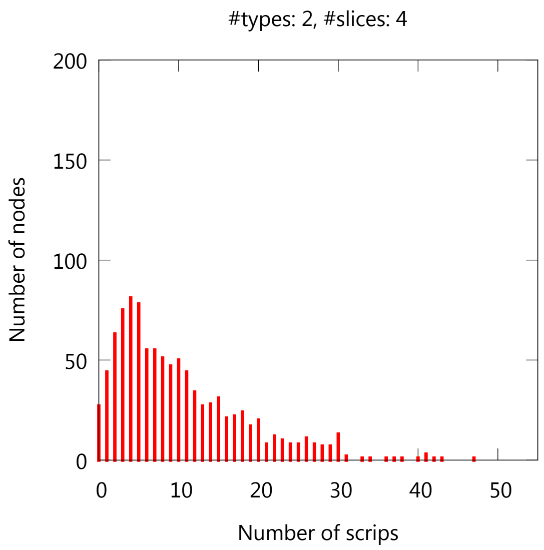


Fig. 5: Scrip distribution after 1,000,000 transactions

3.2 Social Welfare

We are interested in the effects of thresholds, number of slices, and average amount of money on the social welfare. Figure 6 shows that when number of slices increases, social welfare decreases logarithmically. This simulation is run with the following choice of parameters (with slices ranging from 1 to 10):

```
Number of nodes: 1000
Length of the x-coordinate: 200
Length of the y-coordinate: 200
Coverage radius of a node: 0
Average amount of money: 10
Number of transactions: 1000000
Output frequency: 10000
Types: 2

(Type 1)
Ratio: 0.3
Threshold: 30
Alpha: 0.05
Beta: 1
```

```
Gamma: 1
Delta: 0.95
Rho:1

(Type 2)
Ratio: 0.7
Threshold: 40
Alpha: 0.15
Beta: 1
Gamma: 1
Delta: 0.95
Rho:1
```

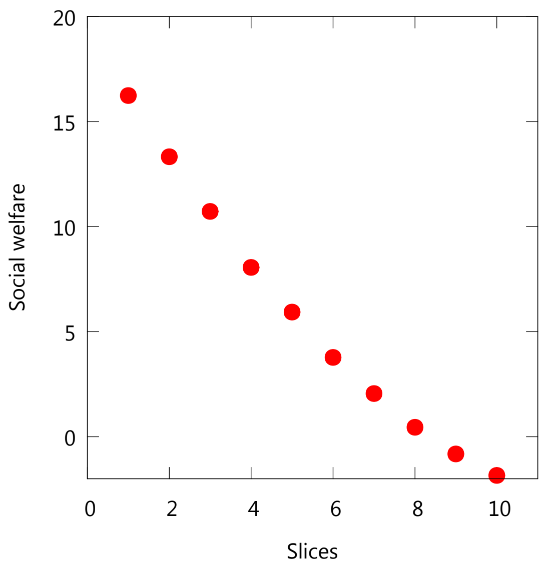


Fig. 6: Social welfare for various amounts of slices

In Figure 7, we show the effect of average amount of money on social welfare. This simulation is run with the following choice of parameters (with average amount of money ranging from 2 to 18):

```
Number of nodes: 1000
Length of the x-coordinate: 200
Length of the y-coordinate: 200
```

```
Coverage radius of a node: 0
Number of slices: 1
Number of transactions: 1000000
Output frequency: 10000
Types: 1

(Type 1)
Ratio: 1
Threshold: 20
Alpha: 0.05
Beta: 1
Gamma: 1
Delta: 0.95
Rho:1
```

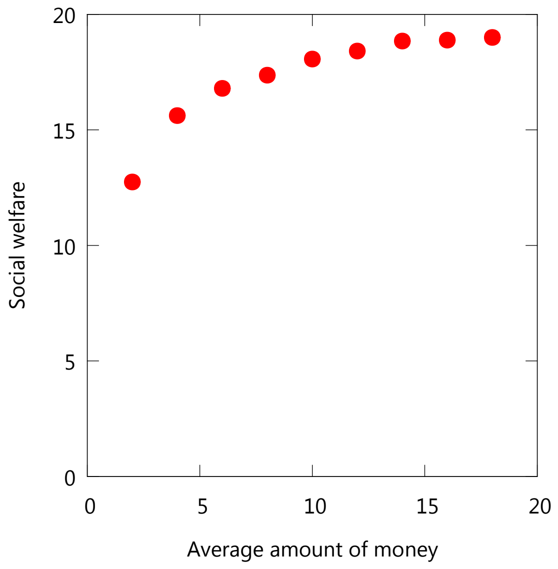


Fig. 7: Social welfare for various amounts of average money

In Figure 8, we show the effect of threshold on social welfare. This simulation is run with the following choice of parameters (with threshold ranging from 5 to 16):

```
Number of nodes: 1000
```

```
Length of the x-coordinate: 200
Length of the y-coordinate: 200
Coverage radius of a node: 0
Number of slices: 1
Average amount of money: 4
Number of transactions: 1000000
Output frequency: 10000
Types: 1

(Type 1)
Ratio: 1
Alpha: 0.05
Beta: 1
Gamma: 1
Delta: 0.95
Rho:1
```

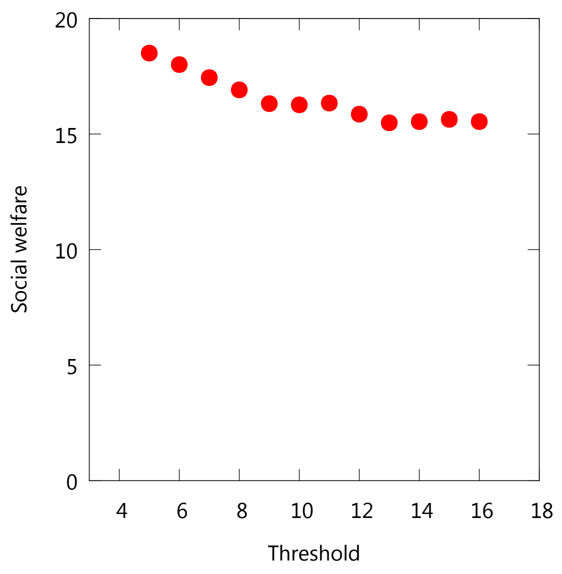


Fig. 8: Social welfare for various amounts of thresholds

4 Conclusion

We would like to adapt scrip system to the privacy-preserving scheme proposed in [1] for participatory sensing. For this reason, we implemented a scrip system simulation infrastructure with many capabilities using the C programming language and we used the gnuplot software to plot graphs from simulation data. We provided simulation results for the money distribution and its relation with the number of slices when the system is run for a long time. Moreover, we provided simulation results about the effects of slices, average amount of money, and thresholds on the social welfare.

As a future work, we would like to add new capabilities to our simulation infrastructure. In the current model, the nodes are fixed in the XY-plane but nodes should be allowed to move since this scrip system is proposed for mobile devices. Moreover, currently nodes can send their slices only to their neighbours but we would like to allow h-hop slice transfer and see the effects of h-hop on the money distribution and the social welfare.

References

1. Shi, J., Zhang, R., Liu, Y., Zhang, Y.: PriSense: Privacy-Preserving Data Aggregation in People-Centric Urban Sensing Systems. In: Proceedings of IEEE INFOCOM 2010, IEEE (March 2010) 1–9
2. Friedman, E.J., Halpern, J.Y., Kash, I.: Efficiency and nash equilibria in a scrip system for P2P networks. In: Proceedings of the 7th ACM conference on Electronic commerce. EC '06, New York, NY, USA, ACM (2006) 140–149
3. Kash, I.A., Friedman, E.J., Halpern, J.Y.: Optimizing scrip systems: efficiency, crashes, hoarders, and altruists. In: Proceedings of the 8th ACM conference on Electronic commerce. EC '07, New York, NY, USA, ACM (2007) 305–315
4. Williams, T., Kelley, C.: gnuplot, http://www.gnuplot.info/

A scrip.c File

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#define max_types 100 // Maximum number of types that can be simulated
#define number_nodes 10000 // Maximum number of nodes that can be simulated
#define max_slices 100 // Maximum number of slices that a data package contains
int distance; // Maximum distance between the nodes (Coverage radius)
int x,y; // Size of the area on x-y plane. There can be at most x*y nodes
int nodes[number_nodes][2]; // Shows the coordinates of the nodes eg: For N5 (x,y)=(nodes[5][0],nodes[5][1])
int neighbours[number_nodes][number_nodes]; // eg: 2nd neigbour of N6 is neighbours[6][1].
int n_neighbours[number_nodes]; // Number of neighbours of a node. eg: N6 has n_neighbours[6] neighbours
int scrip[number_nodes]; // Money of the nodes
int n_transactions=0; // Number of successful data transmission
int history[number_nodes]; // Keeps a track of the nodes that the data sent (to avoid not to send more than 1 data to single node)
int available_nodes; // number of neighbours that are not contacted yet
int minimum_n_neighbours; // The mininum number of neighbours a node has
int total_rejections=0;
int n_brokes; // Number of nodes whose scrip is less than slices
int outputs=0; // Counter for the outputs
int slices; // Number of slices a package contains (fixed) (if changed, also change the array size of 'acknowledged')
int acknowledged[max_slices]; // Keeps note of the nodes that agreed to receive slice
int randomization[100000]; // Keeps the list of nodes for random selection. The array size is number_nodes x upper_bound_data
int modulo=0; // When determining the node to send data, random numbers are generated modulo this number. (i.e. rand()%modulo) See "prepare_table_random_selection" function
int n_rich; // Number of nodes that have scrips higher than or equal to their thresholds
```

```
double ratio[max_types],alpha[max_types],beta[max_types],gamma
     [max_types],delta[max_types],rho[max_types];
int tthreshold[max_types],nnodes; // nnodes: number of nodes,
     tthresholds: threshold of a type
double average_gamma, average_alpha, average_delta;
int m; // average amount of money
int types; // number of types
int transaction,output_frequency; // Simulation length and
     output frequency
int ntype[number_nodes]; // type of each node
FILE *fp,*fp2,*fp3,*fp4,*fp5;
#include "scrip.h"

void define_nodes() { // Radomly creates "nnodes" many nodes
     on the x-y plane
        int i,j,flag;
        for (i=0;i<nnodes;i++) {
                flag=1;
                while(flag) {
                        flag=0;
                        nodes[i][0]=rand()%x;
                        nodes[i][1]=rand()%y;
                        for (j=0;j<i-1;j++) {
                                if (nodes[j][0]==nodes[i][0]
                                    && nodes[j][1]==nodes[i
                                    ][1]) flag=1;
                        }
                }
        }
}
void prepare_variables() { // Initializes some values to zero
        int i;
        for (i=0;i<nnodes;i++) {n_neighbours[i]=0;}
        fp3 = fopen("scrips.txt", "w");
        fp4 = fopen("social_welfare.txt", "w");
        fclose(fp3);
        fclose(fp4);
}
void detect_neighbours() { // Every node recognizes its
     neighbours
        int i,j;
        int temp1,temp2,temp3;
        temp3=distance*distance;
        for (i=0;i<nnodes;i++) {
                for (j=0;j<nnodes;j++) {
                        if (j!=i) {
                                temp1=nodes[i][0]-nodes[j][0];
                                    temp1*=temp1;
                temp2=nodes[i][1]-nodes[j][1]; temp2=temp2*temp2+temp1;
                if (temp2<=temp3) {
                    neighbours[i][n_neighbours[i]]=j;
                    n_neighbours[i]++;
                }
            }
        }
    }
    minimum_n_neighbours=n_neighbours[0];
    for (i=1;i<nnodes;i++) if (minimum_n_neighbours>n_neighbours[i]) minimum_n_neighbours=n_neighbours[i];
    if (minimum_n_neighbours==0) {
        printf("There is a node that does not have any neighbours\n");
        exit(1);
    }
}
int check_consistency() { // Checks if the user's input values make sense
    int flag=1,i;
    double total_ratio=0;
    average_gamma=0; average_alpha=0; average_delta=0;
    for (i=0;i<types;i++) {
        if (beta[i]<0 || beta[i]>1) {printf("Beta value incorrect\n"); exit(1);}
        if (ratio[i]<0 || ratio[i]>1) {printf("Ratio value incorrect\n"); exit(1);}
        total_ratio+=ratio[i];
        average_gamma+=gamma[i]*ratio[i];
        average_alpha+=alpha[i]*ratio[i];
        average_delta+=delta[i]*ratio[i];
    }
    printf("average_gamma: %lf average_alpha: %lf average_delta:%lf\n",average_gamma,average_alpha,average_delta);
    if (total_ratio!=1) {printf("Ratios do not sum to 1, it is %lf\n",total_ratio);exit(1);}
    if (nnodes>x*y) {printf("X-Y plane is too small for this many nodes\n"); exit(1);}
    return flag;
}
void distribute_money() { // Amount of initial scrip is described here
    int i,j,temp1,temp2,min=0;
    temp1=m/2;
```

```
        temp2=m+m-temp1;
        for (i=0;i<types;i++) {
                for (j=min;j<min+ratio[i]*nnodes;j++) {
                        ntype[j]=i;
                        if (j%2==0) scrip[j]=temp1;
                        else scrip[j]=temp2;
                }
                min=j;
        }
}
int volunteers() {
        int i;
        for (i=0;i<nnodes;i++) {
                if (history[i]==0) return 1;
        }
        return 0;
}
void send_data(int requester) {// Ni sends slices to acknowledged nodes (Scrips are altered and behaviours may change)
        int i;
        scrip[requester]-=slices;
        for (i=0;i<slices;i++) scrip[acknowledged[i]]++;
        n_transactions++;
        if (n_transactions%output_frequency==0) {record_scrips(); record_social_welfare();}
}

int acknowledge_transmission(int requester) {
        int ack=0,i,volunteer;
        double temp;
        for (i=0;i<nnodes;i++) history[i]=0;
        history[requester]=1;
        while (volunteers() && ack<slices) {
                volunteer=rand()%nnodes;
                if (scrip[volunteer]<tthreshold[ntype[volunteer]]) {
                        if (history[volunteer]==0) {
                                temp=rand();
                                temp/=32768;
                                if (temp<=beta[ntype[volunteer]]) {
                                        acknowledged[ack]=volunteer;
                                        ack++;
                                }
                        }
                }
                history[volunteer]=1;
        }
        if (ack==slices) return 1;
        else total_rejections++;
        return 0;
}
void exchange_data() {
        int requester, flag;
        requester=rand()%modulo;
        requester=randomization[requester];
        if (scrip[requester]>=slices) {
                flag=acknowledge_transmission(requester);
                if (flag) send_data(requester);
        }
}
void prepare_table_random_selection() {
        int i,j;
        for (i=0;i<nnodes;i++) {
                for (j=0;j<rho[ntype[i]];j++) {
                        randomization[modulo]=i;
                        modulo++;
                }
        }
}
void calculate_brokes() {
        int i;
        n_brokes=0; n_rich=0;
        for (i=0;i<nnodes;i++) {
                if (scrip[i]<slices) n_brokes++;
                else if (scrip[i]>=tthreshold[ntype[i]])
                    n_rich++;
        }
}
void input_parameters() {
        int i;
        printf("Number of nodes (maximum: 10000): ");
        scanf("%d",&nnodes);
        printf("Length of the x-coordinate: ");
        scanf("%d",&x);
        printf("Length of the y-coordinate: ");
        scanf("%d",&y);
        printf("Coverage radius of a node (0: unlimited): ");
        scanf("%d",&distance);
        if (distance==0) distance=x*y;
        printf("Number of slices: ");
        scanf("%d",&slices);
        printf("Average amount of money: ");
        scanf("%d",&m);
        printf("Number of transactions (default: 1000000): ");
        scanf("%d",&transaction);
        printf("Output frequency (default: 10000): ");
        scanf("%d",&output_frequency);
        printf("Number of types: ");
        scanf("%d",&types);
        for (i=0;i<types;i++) {
                printf("\n\t (Type %d)\n",i+1);
                printf("Ratio: ");
                scanf("%lf",&ratio[i]);
                printf("Threshold: ");
                scanf("%d",&tthreshold[i]);
                printf("Alpha (Cost of satisfying a request):
                    ");
                scanf("%lf",&alpha[i]);
                printf("Beta (Probability of satisfying a
                    request): ");
                scanf("%lf",&beta[i]);
                printf("Gamma (Utility the agent gains): ");
                scanf("%lf",&gamma[i]);
                printf("Delta (Rate at which an agent
                    discounts utility): ");
                scanf("%lf",&delta[i]);
                printf("Rho (Relative request rate): ");
                scanf("%lf",&rho[i]);
        }
        fprint_parameters();
}
void main () {
        srand((unsigned int)time(NULL)); // seed the PRNG with
             the current time
        input_parameters();
        check_consistency();
        prepare_variables();
        define_nodes();
        detect_neighbours();
        distribute_money();
        record_scrips();
        prepare_table_random_selection();
        while (n_transactions<=transaction) {
                exchange_data();
                calculate_brokes();
        }
}
```

B scrip.h File

```
void print_scrip() {
        int i;
        for (i=0;i<nnodes;i++) {
                printf("Scrip of Node %d: \t%d\n",i,scrip[i]);
        }
        printf("Number of transactions: %d\n",n_transactions);
        printf("Number of rejections: %d\n",total_rejections);
}
void print_n_neighbours() {
        int i;
        for (i=0;i<nnodes;i++) printf("#Neighbours of %d: \t%d
            \n",i,n_neighbours[i]);
        printf ("Minimum number of neighbours: %d\n",
            minimum_n_neighbours);

}
void print_neighbours () {
        int i,j;
        for (i=0;i<nnodes;i++) {
                printf("Neighbours of N%d: ",i);
                for (j=0;j<n_neighbours[i];j++) {
                        printf("N%d ",neighbours[i][j]);
                }
                printf("\n");
        }
}
void print_nodes() {
        int i,j;
        for (i=0;i<nnodes;i++) {
                printf("Node %d:\t\t",i);
                for (j=0;j<2;j++) {
                        printf ("%d\t",nodes[i][j]);
                }
                printf("\n");
        }
}
void fprint_nodes() {
        int i;
        fp = fopen("nodes.txt", "w");
        fp2 = fopen("gnu_nodes.txt", "w");
        for (i=0;i<nnodes;i++) {
                fprintf(fp,"%d %d %d N%d\n",nodes[i][0],nodes[
                    i][1],distance,i);
        }
        fprintf(fp2,"set term pdf size 3.00in, 3.00in\n");
        fprintf(fp2,"set output \'nodes.pdf\'\n");
        fprintf(fp2,"set size ratio 1\n");
```

```c
        fprintf(fp2,"set pointsize 0.5\n");
        fprintf(fp2,"set xrange[-10:%d]\n",x+10+distance);
        fprintf(fp2,"set yrange[-10:%d]\n",y+10+distance);
        fprintf(fp2,"plot \'nodes.txt\' using 1:2:3 ti 'Coverage' with circles, \'nodes.txt\' using (\$1+3):2:4 ti '' with labels font \"Verdana, 3\", \'nodes.txt\' using 1:2 ti 'Nodes' with points pt 7 lc rgb \"black\"");
        fclose(fp);
        fclose(fp2);
}
void record_scrips() {
        int i,j,scrip_total=0,count;
        fp3 = fopen("scrips.txt", "ab");
        for (i=0;i<nnodes;i++) scrip_total+=scrip[i];
        fprintf(fp3,"# %d\n",outputs); outputs++;
        for (i=0;i<=scrip_total;i++) {
                count=0;
                for (j=0;j<nnodes;j++) {
                        if (scrip[j]==i) count++;
                }
                if (count) fprintf(fp3,"%d %d\n",i,count);
        }
        fprintf(fp3,"\n\n");
        fclose(fp3);
}
void record_social_welfare() {
        int temp;
        double social_welfare;
        fp4 = fopen("social_welfare.txt", "ab");
        social_welfare=n_brokes;
        social_welfare=(1-social_welfare/nnodes)*(average_gamma-slices*average_alpha)/(1-average_delta);
        temp=n_transactions/output_frequency;
        fprintf(fp4,"%d %lf\n",temp,social_welfare );
        fclose(fp4);
}
void fprint_parameters() {
        int i;
        fp5 = fopen("parameters.txt", "w");
        fprintf(fp5,"Number of nodes: %d\n",nnodes);
        fprintf(fp5,"Length of the x-coordinate: %d\n",x);
        fprintf(fp5,"Length of the y-coordinate: %d\n",y);
        fprintf(fp5,"Coverage radius of a node (0: unlimited): %d\n",distance);
        fprintf(fp5,"Number of slices: %d\n",slices);
        fprintf(fp5,"Average amount of money: %d\n",m);
        fprintf(fp5,"Number of transactions: %d\n",transaction
            );
        fprintf(fp5,"Output frequency: %d\n",output_frequency)
            ;
        fprintf(fp5,"Number of types: %d\n",types);
        for (i=0;i<types;i++) {
                fprintf(fp5,"\n\t (Type %d)\n",i+1);
                fprintf(fp5,"Ratio: %lf\n",ratio[i]);
                fprintf(fp5,"Threshold: %d\n",tthreshold[i]);
                fprintf(fp5,"Alpha: %lf\n",alpha[i]);
                fprintf(fp5,"Beta: %lf\n",beta[i]);
                fprintf(fp5,"Gamma: %lf\n",gamma[i]);
                fprintf(fp5,"Delta: %lf\n",delta[i]);
                fprintf(fp5,"Rho: %lf\n",rho[i]);
        }
        fclose(fp5);
}
```