Simulations for Verifying the Differential Privacy of Gossip Protocols

# Background

Differential privacy is used to measure the capability of a randomized algorithm that protects the data from being identified in data science. This notion is extended to the area of information spreading, where the differential privacy is used to measure the capability of a randomized algorithm (an information spreading protocol in this case) that protects the information source from being identified. Theoretical proofs for the differential privacy of general gossip protocols is provided in the paper. To further verify these theoretical results, simulations are needed.

# Problem Description

The underlying problem includes three parts: information spreading, gossip protocols, and network models. First, information spreading is a random process that starts with an initial node with the information and information will be disseminated to all nodes with the given information spreading protocol. This study focuses on the gossip protocol, a typical information spreading protocol. The core of the gossip protocol is the so-called gossip step. During the gossip-based information spreading, if a node is activated and it will perform the gossip action, that is, uniformly randomly chose a neighbor node and push (here we focus on the push-based gossip protocol) the information it has to the chosen neighbor. So, if this activated node has no information, nothing happens; but if this active node has the information, the chosen node will also have the information after the gossip step. Check the code for more details. Another important part of the gossip protocol is the time model. The time model describes when nodes are activated. There are two types of time models: synchronous and asynchronous time model. In the synchronous time model, all nodes will be activated at a discrete-time axis and perform the gossip action. So, it is quite straightforward, just check the synchronous gossip function in the code. The other one is the asynchronous time model, in this model, nodes have their own clock that ticks following a Poisson random process with a given rate (here in our simulations the rate is 1). Equivalently, that means, the time interval between two consecutive ticks is a random value following the exponential distribution with the given rate (i.e., 1). In the code, this time interval is generated by the PoissonSample function, you just need to input the rate and it will generate the random time interval. When a node’s clock ticks, it is activated and will perform the gossip action. The whole asynchronous time model is simulated through a time heap (heap is a data structure that automatically arranges the elements in a list based on this value). The spreading process is mimicked through the heap popping the next most recent event and updates itself with the outcome from this event. Again, check the EstimateAsynchronousGossipTime function to understand this part. Finally, it is the network model. Here in this study, we consider two random network models: Erdos Renyi (ER) network and Geometric Random (GR) network. ER network is a network that a connection is established between any two nodes in the network with a probability of p. GR network is a network that nodes are uniformly randomly placed in a 2-d 1by1 space, each node connects to all other nodes that have a Euclidean distance of r from it. So, ER network is constructed through parameters n (number of nodes) and p, while GR network is constructed through n and r. Check the code for more details of how these two networks are constructed.

# Works to Be Done

1. **Implement the function that can simulate the private gossip.**

The private gossip has an additional “muting” parameter . Specifically, each time after an informed and activated node performs the gossip action, it has a probability of that will turn muted, i.e., it can no longer be activated to perform the gossip action, unless this node is informed by another node again. In the pseudo code for asynchronous gossip, it shall be like the following:

event\_heap.pop() -> next activated node

-> node in Neighbor[]

performs the gossip action to

if rand() < s:

heappush(event\_heap, the next event time of )

heappush(event\_heap, the next event time of )

1. **Implement the monitoring and predicting functions of the attacker.**

Assume that the attacker can monitor all communications happening during the information spreading but with a success probability of , i.e., with a probability of the attacker knows who initiates the gossip action in the current time. Also, assumes that the source node belongs to a subset of all nodes , which is known by the attacker, it then predicts the first node it observes that belongs to this subset as the source node. Compare this prediction with the actual source to see if it correctly identifies the source. So in each simulation, you need to perform the following step:

1. Select the portion of candidate nodes for the source. For example, 10% of the nodes can be the source node, then uniformly pick 10% nodes from the total nodes as the candidates. These nodes will be the candidates for the actual source and also the knowledge for the attacker to infer the source. Ultimately, you need to chose different value of portions. The suggestion is you pick [1%, 10%, 50%, 100%].
2. At each simulation, randomly choose one node from as the source and starts the gossip-based spreading process. Meanwhile, the attacker starts monitoring. Once it monitors any activity from a node (e.g., ) in for the first time, the attacker decides should be the source node. Stops simulating the gossip-based spreading process, compare this node with the actual source to find out if the prediction is correct or not.
3. Run the simulation with enough repetitions (i.e., Monte-Carlo runs), the corresponding prediction accuracy can be obtained and this will be used to reflect the privacy guarantees of the private gossip protocol in different scenarios. If given a fixed simulation environment (i.e., all parameters are fixed except for the portion of candidate nodes), when the portion of candidate nodes changes, the more the prediction accuracy changes, the worse differential privacy is guaranteed. Refere to the Fig. 3(a) in [1] for more reference.

The monitoring part can be directly added to the above private gossip function like below:

event\_heap.pop() -> next activated node

monitored\_nodes <- []

chose node in Neighbor[]

performs the gossip action to

if rand() < :

monitored\_nodes.append()

if rand() < s:

heappush(event\_heap, the next event time of )

heappush(event\_heap, the next event time of )

1. **Perform the simulations for different factors that can impact the differential privacy.**

Simulate the prediction accuracy for **the network diameter , the success probability , the wireless connection fail probability , the muting probability s, and the delay time .** The network diameter can be adjusted through changing the average number of neighbors of network generated. For both the ER and GR networks with a fixed number of nodes, if you increase the average number of neighbors (increase p for ER and increase r for GR), the diameter will decrease, vice versa. The diameter can then be calculated through the networkx.diameter (or nx.diameter(G) in the code) function. The wireless connection fail probability is another parameter to be simulated; this is the probability that the node successfully performs the gossip action. In the above code, you can just add one line like below to achieve this.

event\_heap.pop() -> next activated node

monitored\_nodes <- []

chose node in Neighbor[]

**if rand() < f:**

performs the gossip action to

if rand() < :

monitored\_nodes.append()

if rand() < s:

heappush(event\_heap, the next event time of )

heappush(event\_heap, the next event time of )

This is also used to simulate the corresponding spreading time for the information spreading process. Check the code for how to estimate the spreading time for the original gossip without the fail probability. The delay time is the time when the attacker starts to monitor the spreading. So you need to add a counter in the above code like below:

**counter <- 0**

while true:

event\_heap.pop() -> next activated node

monitored\_nodes <- []

chose node in Neighbor[]

if rand() < f:

performs the gossip action to

**If counter > t and rand() < :**

monitored\_nodes.append()

if rand() < s:

heappush(event\_heap, the next event time of )

heappush(event\_heap, the next event time of )

**counter += 1**

So, the tasks to be done for this part include:

1. (Sida) Plot the prediction accuracy curves for different values of , , s, and t. In all, four curves in total. Plot the similar figure as Fig. 3(a) in [1] for each of these parameters.
2. (Jingzhe) Plot one figure for the prediction accuracy for different values of wireless connection fail probability , similar as as Fig. 3(a) in [1] with the x-axis to be replaced with . Plot another figure with different spreading times for different values of

**Reference:**

[1] Bellet, A., Guerraoui, R., & Hendrikx, H. (2019). Who started this rumor? Quantifying the natural differential privacy guarantees of gossip protocols. *arXiv preprint arXiv:1902.07138*.