Communication-Efficient Triangle Counting under Local Differential Privacy

Jacob Imola, *UC San Diego;* Takao Murakami, *AIST;*
Kamalika Chaudhuri, *UC San Diego*

https://www.usenix.org/conference/usenixsecurity22/presentation/imola

This paper is included in the Proceedings of the
31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Proceedings of the
31st USENIX Security Symposium is
sponsored by USENIX.

Communication-Efficient Triangle Counting under Local Differential Privacy

Jacob Imola* *UC San Diego*
Takao Murakami* *AIST*
Kamalika Chaudhuri *UC San Diego*

Abstract

Triangle counting in networks under LDP (Local Differential Privacy) is a fundamental task for analyzing connection patterns or calculating a clustering coefficient while strongly protecting sensitive friendships from a central server. In particular, a recent study proposes an algorithm for this task that uses two rounds of interaction between users and the server to significantly reduce estimation error. However, this algorithm suffers from a prohibitively high communication cost due to a large noisy graph each user needs to download.

In this work, we propose triangle counting algorithms under LDP with a small estimation error and communication cost. We first propose two-rounds algorithms consisting of edge sampling and carefully selecting edges each user downloads so that the estimation error is small. Then we propose a double clipping technique, which clips the number of edges and then the number of noisy triangles, to significantly reduce the sensitivity of each user's query. Through comprehensive evaluation, we show that our algorithms dramatically reduce the communication cost of the existing algorithm, e.g., from 6 hours to 8 seconds or less at a 20 Mbps download rate, while keeping a small estimation error.

1 Introduction

Counting subgraphs (e.g., triangles, stars, cycles) is one of the most basic tasks for analyzing connection patterns in various graph data, e.g., social, communication, and collaboration networks. For example, a triangle is given by a set of three nodes with three edges, whereas a k-star is given by a central node connected to k other nodes. These subgraphs play a crucial role in calculating a *clustering coefficient* ($= \frac{3\times\#\text{triangles}}{\#\text{2-stars}}$) (see Figure 1). The clustering coefficient measures the average probability that two friends of a user will also be a friend in a social graph [45]. Therefore, it is useful for measuring the effectiveness of friend suggestions. In addition, the clustering coefficient represents the degree to which users tend to cluster together. Thus, if it is large in some services/communities, we can effectively apply social recommendations [38] to the users. Triangles and k-stars are also useful for constructing graph models [31, 52]; see also [59] for other applications of triangle counting. However, graph data often involve sensitive data such as sensitive edges (friendships), and they can be leaked from exact numbers of triangles and k-stars [29].

To analyze subgraphs while protecting user privacy, DP (Differential Privacy) [25] has been widely adopted as a privacy metric [24, 29, 36, 56, 64, 65, 67]. DP protects user privacy against adversaries with arbitrary background knowledge and is known as a gold standard for data privacy. According to the underlying model, DP can be categorized into *central (or global) DP* and *LDP (Local DP)*. Central DP assumes a scenario where a central server has personal data of all users. Although accurate analysis of subgraphs is possible under this model [24, 36, 67], there is a risk that the entire graph is leaked from the server by illegal access or internal fraud [18, 43]. In addition, central DP cannot be applied to *decentralized social networks* [4–6, 48] where the entire graph is distributed across many servers. We can even consider *fully decentralized applications* where a server does not have any original edge, e.g., a mobile app that sends a noisy degree (noisy number of friends) to the server, which then estimates a degree distribution. Central DP cannot be used in such applications.

In contrast, LDP assumes a scenario where each user obfuscates her personal data (friends list in the case of graphs) by herself and sends the obfuscated data to a possibly malicious server; i.e., it does not assume trusted servers. Thus, it does not suffer from a data breach and can also be applied to the decentralized applications. LDP has been widely studied in tabular data where each row corresponds to a user's personal data (e.g., age, browser setting, location) [8, 12, 27, 33, 44, 60] and also in graph data [29, 49, 64, 65]. For example, k-star counts can be very accurately estimated under LDP because each user can count k-stars of which she is a center and sends a noisy version of her k-star count to the server [29].

However, more complex subgraphs such as triangles are much harder to count under LDP because each user cannot see

*The first and second authors made equal contribution.

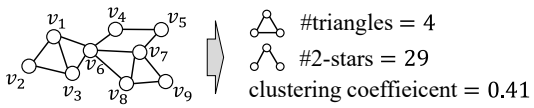


Figure 1: Triangles, 2-stars, and clustering coefficient.

edges between other users. For example, in Figure 1, user v_1 cannot see edges between v_2, v_3, and v_6 and therefore cannot count triangles involving v_1. Thus, existing algorithms [29, 64, 65] obfuscate each user's edges (rather than her triangle count) by RR (Randomized Response) [61] and send noisy edges to a server. Consequently, the server suffers from a prohibitively large estimation error (e.g., relative error $> 10^2$ in large graphs, as shown in Appendix B) because all three edges are noisy in any noisy triangle the server sees.

A recent study [29] shows that the estimation error in locally private triangle counting is significantly reduced by introducing an additional round of interaction between users and the server. Specifically, if the server publishes the noisy graph (all noisy edges) sent by users at the first round, then each user can count her noisy triangles such that *only one edge* is noisy (as she knows two edges connected to her). Thus, the algorithm in [29] sends each user's noisy triangle count (with additional noise) to the server at the second round. Then the server can accurately estimate the triangle count. This algorithm also requires a much smaller number of interactions (i.e., only two) than collaborative approaches [34, 55] that generally require many interactions.

Unfortunately, the algorithm in [29] is still impractical for a large-scale graph. Specifically, the noisy graph sent by users is dense, hence extremely large for a large-scale graph, e.g., 500 Gbits for a graph of a million users. The problem is that *every user* needs to download such huge data; e.g., when the download speed is 20 Mbps (which is a recommended speed in YouTube [7]), every user needs about 7 hours to download the noisy graph. Since the communication ability might be limited for some users, the algorithm in [29] cannot be used for applications with large and diverse users.

In summary, existing triangle algorithms under LDP suffer from either a prohibitively large estimation error or a prohibitively high communication cost. They also suffer from the same issues when calculating the clustering coefficient.

Our Contributions. We propose locally private triangle counting algorithms with a small estimation error and small communication cost. Our contributions are as follows:

- We propose two-rounds triangle algorithms consisting of *edge sampling* after RR and *selecting edges each user downloads*. In particular, we show that a simple extension of [29] with edge sampling suffers from a large estimation error for a large or dense graph where the number of 4-cycles (such as v_1-v_2-v_3-v_6-v_1 in Figure 1) is large. To address this issue, we propose some strategies for selecting edges to download to reduce the error caused by the 4-cycles, which we call the *4-cycle trick*.
- We show that the algorithms with the 4-cycle trick still suffer from a large estimation error due to large Laplacian noise for each user. To significantly reduce the Laplacian noise, we propose a *double clipping* technique, which clips a degree (the number of edges) of each user with LDP and then clips the number of noisy triangles.
- We evaluate our algorithms using two real datasets. We show that our entire algorithms with the 4-cycle trick and double clipping dramatically reduce the communication cost of [29]. For example, for a graph with about 900000 users, we reduce the download cost from 400 Gbits (6 hours when 20 Mbps) to 160 Mbits (8 seconds) or less while keeping the relative error much smaller than 1.

Thus, locally private triangle counting is now much more practical. In Appendix C, we also show that we can estimate the clustering coefficient with a small estimation error and download cost. For example, our algorithms are useful for measuring the effectiveness of friend suggestions or social recommendations in decentralized social networks, e.g., Diaspora [4], Mastodon [5]. Our source code is available at [1].

All the proofs of our privacy and utility analysis are given in the full version [30].

Technical Novelty. Below we explain more about the technical novelty of this paper. Although we focus on two-rounds local algorithms in the same way as [29], we introduce several new algorithmic ideas previously unknown in the literature.

First, our 4-cycle trick is totally new. Although some studies focus on 4-cycle counting [13, 35, 40, 42], this work is the first to use 4-cycles to improve communication efficiency. Second, selective download of parts of a centrally computed quantity is also new. This is not limited to graphs – even in machine learning, there are no such strategic download techniques previously, to our knowledge. Third, our utility analysis of our triangle algorithms (Theorem 2) is different from [29] in that ours introduces subgraphs such as 4-cycles and k-stars. This leads us to our 4-cycle trick. Fourth, we propose two triangle algorithms that introduce the 4-cycle trick and show that the more tricky one provides the best performance because of its low sensitivity in DP.

Finally, our double clipping is new. Andrew *et al.* [9] propose an adaptive clipping technique, which applies clipping twice. However, they focus on federated averaging, and their problem setting is different from our graph setting. In particular, they require a private quantile of the norm distribution. In contrast, we need only a much simpler estimate: a private degree. Here, we use the fact that the degree has a small sensitivity (sensitivity $= 1$) in DP for edges. We also provide a new, reasonably tight bound on the probability that the noisy triangle count exceeds a clipping threshold (Theorem 4). Thanks to the two differences, we obtain a significant communication improvement: two or three orders of magnitude.

2 Related Work

Triangle Counting. Triangle counting has been extensively studied in a non-private setting [14, 15, 21, 26, 54, 57, 58, 62] (it is almost a sub-field in itself) because it requires high time complexity for large graphs.

Edge sampling [14, 26, 58, 62] is one of the most basic techniques to improve scalability. Although edge sampling is simple, it is quite effective – it is reported in [62] that edge sampling outperforms other sampling techniques such as node sampling and triangle sampling. Based on this, we adopt edge sampling after RR[1] with new techniques such as the 4-cycle trick and double clipping. Our entire algorithms significantly improve the communication cost, as well as the space and time complexity, under LDP (see Sections 5.3 and 6).

DP on Graphs. For private graph analysis, DP has been widely adopted as a privacy metric. Most of them adopt central (or global) DP [23, 24, 28, 36, 37, 50, 67], which suffers from the data breach issue.

LDP on graphs has recently studied in some studies, e.g., synthetic data generation [49], subgraph counting [29, 56, 64, 65]. A study in [56] proposes subgraph counting algorithms in a setting where each user allows her friends to see all her connections. However, this setting is unsuitable for many applications; e.g., in Facebook, a user can easily change her setting so that her friends cannot see her connections.

Thus, we consider a model where each user can see only her friends. In this model, some one-round algorithms [64,65] and two-rounds algorithms [29] have been proposed. However, they suffer from a prohibitively large estimation error or high communication cost, as explained in Section 1.

Recently proposed network LDP protocols [22] consider, instead of a central server, collecting private data with user-to-user communication protocols along a graph. They focus on sums, histograms, and SGD (Stochastic Gradient Descent) and do not provide subgraph counting algorithms. Moreover, they focus on hiding each user's private dataset rather than hiding an edge in a graph. Thus, their approach cannot be applied to our task of subgraph counting under LDP for edges. The same applies to another work [53] that improves the utility of an averaging query by correlating the noise of users according to a graph.

LDP. RR [33, 61] and RAPPOR [27] have been widely used for tabular data in LDP. Our work uses RR in part of our algorithm but builds off of it significantly. One noteworthy result in this area is HR (Hadamard Response) [8], which is state-of-the-art for tabular data and requires low communication. However, this result is not applied to graph data and does not address the communication issues considered in this paper. Specifically, applying HR to each bit in a neighbor list will result in $O(n^2)$ (n: #users) download cost in the same way as the previous work [29] that uses RR. Applying HR to an entire neighbor list (which has 2^n possible values) will similarly result in $O(n \log 2^n) = O(n^2)$ download cost.

Previous work on distribution estimation [33, 44, 60] or heavy hitters [12] addresses a different problem than ours, as they assume that every user has i.i.d. (independent and identically distributed) samples. In our setting, a user's neighbor list is non-i.i.d. (as one edge is shared by two users), which does not fit into their statistical framework.

[1] We also note that a study in [46] proposes a graph publishing algorithm in the central model that independently changes 1-cells (edges) to 0-cells (no edges) with some probability and then changes a fixed number of 0-cells to 1-cells *without replacement*. However, each 0-cell is *not* independently sampled in this case, and consequently, their proof that relies on the independence of the noise to each 0-cell is incorrect. In contrast, our algorithms provide DP because we apply sampling after RR, i.e., post-processing.

3 Preliminaries

3.1 Notations

We begin with basic notations. Let $\mathbb{N}$, $\mathbb{R}$, $\mathbb{Z}_{\geq 0}$, and $\mathbb{R}_{\geq 0}$ be the sets of natural numbers, real numbers, non-negative integers, and non-negative real numbers, respectively. For $z \in \mathbb{N}$, let $[z]$ a set of natural numbers from 1 to z; i.e., $[z] = \{1, 2, \ldots, z\}$.

Let $G = (V, E)$ be an undirected graph, where V is a set of nodes and $E \subseteq V \times V$ is a set of edges. Let $n \in \mathbb{N}$ be the number of nodes in V. Let $v_i \in V$ be the i-th node; i.e., $V = \{v_1, \ldots, v_n\}$. We consider a social graph where each node in V represents a user and an edge $(v_i, v_j) \in E$ represents that v_i is a friend with v_j. Let $d_{max} \in \mathbb{N}$ be the maximum degree of G. Let $\mathcal{G}$ be a set of graphs with n nodes. Let $f_\triangle : \mathcal{G} \to \mathbb{Z}_{\geq 0}$ be a triangle count query that takes $G \in \mathcal{G}$ as input and outputs a triangle count $f_\triangle(G)$ (i.e., number of triangles) in G.

Let $\mathbf{A} = (a_{i,j}) \in \{0,1\}^{n \times n}$ be a symmetric adjacency matrix corresponding to G; i.e., $a_{i,j} = 1$ if and only if $(v_i, v_j) \in E$. We consider a local privacy model [29, 49], where each user obfuscates her *neighbor list* $\mathbf{a}_i = (a_{i,1}, \ldots, a_{i,n}) \in \{0,1\}^n$ (i.e., the i-th row of $\mathbf{A}$) using a *local randomizer* $\mathcal{R}_i$ with domain $\{0,1\}^n$ and sends obfuscated data $\mathcal{R}_i(\mathbf{a}_i)$ to a server. We also assume a two-rounds algorithm in which user v_i downloads a message M_i from the server at the second round.

We also show the basic notations in Table 2 of Appendix A.

3.2 Local Differential Privacy on Graphs

LDP on Graphs. When we apply LDP (Local DP) to graphs, we follow the direction of *edge DP* [47, 51] that has been developed for the central DP model. In edge DP, the existence of an edge between any two users is protected; i.e., two computations, one using a graph with the edge and one using the graph without the edge, are indistinguishable. There is also another privacy notion called *node DP* [28, 66], which hides the existence of one user along with all her edges. However, in the local model, many applications send a user ID to a server; e.g., each user sends the number of her friends along with her user ID. For such applications, we cannot use node DP but can use edge DP to hide her edges, i.e., friends. Thus,

we focus on edge DP in the local model in the same way as [29, 49, 56, 64, 65].

Specifically, assume that user v_i uses her local randomizer $\mathcal{R}_i$. We assume that the server and other users can be honest-but-curious adversaries and that they can obtain all edges except for user v_i's edges as prior knowledge. Then we use the following definition for $\mathcal{R}_i$:

Definition 1 (ε-edge LDP [49])**.** *Let $\varepsilon \in \mathbb{R}_{\geq 0}$. For $i \in [n]$, let $\mathcal{R}_i$ be a local randomizer of user v_i that takes $\mathbf{a}_i$ as input. We say $\mathcal{R}_i$ provides* ε-edge LDP *if for any two neighbor lists $\mathbf{a}_i, \mathbf{a}'_i \in \{0,1\}^n$ that differ in one bit and any $s \in \mathrm{Range}(\mathcal{R}_i)$,*

$$\Pr[\mathcal{R}_i(\mathbf{a}_i) = s] \leq e^{\varepsilon} \Pr[\mathcal{R}_i(\mathbf{a}'_i) = s]. \tag{1}$$

For example, a local randomizer $\mathcal{R}_i$ that applies Warner's RR (Randomized Response) [61], which flips 0/1 with probability $\frac{1}{e^{\varepsilon}+1}$, to each bit of $\mathbf{a}_i$ provides ε-edge LDP.

The parameter ε is called the privacy budget. When ε is small (e.g., $\varepsilon \leq 1$ [39]), each bit is strongly protected by edge LDP. Edge LDP can also be used to hide *multiple bits* – by group privacy [25], two neighbor lists $\mathbf{a}_i, \mathbf{a}'_i \in \{0,1\}^n$ that differ in $k \in \mathbb{N}$ bits are indistinguishable up to the factor $k\varepsilon$.

Edge LDP is useful for protecting a neighbor list $\mathbf{a}_i$ of each user v_i. For example, a user in Facebook can change her setting so that anyone (except for the central server) cannot see her friend list $\mathbf{a}_i$. Edge LDP hides $\mathbf{a}_i$ even from the server.

As with regular LDP, the guarantee of edge LDP does not break even if the server or other users act maliciously. However, adding or removing an edge affects the neighbor list of two users. This means that each user needs to trust her friend to not reveal an edge between them. This also applies to Facebook – even if v_i keeps $\mathbf{a}_i$ secret, her edge with v_j can be disclosed if v_j reveals $\mathbf{a}_j$. To protect each edge during the whole process, we use another privacy notion called relationship DP [29]:

Definition 2 (ε-relationship DP [29])**.** *Let $\varepsilon \in \mathbb{R}_{\geq 0}$. For $i \in [n]$, let $\mathcal{R}_i$ be a local randomizer of user v_i that takes $\mathbf{a}_i$ as input. We say $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ provides* ε-relationship DP *if for any two neighboring graphs $G, G' \in \mathcal{G}$ that differ in one edge and any $(s_1, \ldots, s_n) \in \mathrm{Range}(\mathcal{R}_1) \times \ldots \times \mathrm{Range}(\mathcal{R}_n)$,*

$$\begin{aligned} &\Pr[(\mathcal{R}_1(\mathbf{a}_1), \ldots, \mathcal{R}_n(\mathbf{a}_n)) = (s_1, \ldots, s_n)] \\ &\leq e^{\varepsilon} \Pr[(\mathcal{R}_1(\mathbf{a}'_1), \ldots, \mathcal{R}_n(\mathbf{a}'_n)) = (s_1, \ldots, s_n)], \end{aligned} \tag{2}$$

where $\mathbf{a}_i$ (resp. $\mathbf{a}'_i$) $\in \{0,1\}^n$ is the i-th row of the adjacency matrix of graph G (resp. G').

If users v_i and v_j follow the protocol, (2) holds for graphs G, G' that differ in (v_i, v_j). Thus, relationship DP applies to all edges of a user whose neighbors are trustworthy.

While users need to trust other friends to maintain a relationship DP guarantee, only one edge per user is at risk for each malicious friend that does not follow the protocol. This is because only one edge can exist between two users. Thus, although the trust assumption in relationship DP is stronger than that of LDP, it is much weaker than that of central DP in which all edges can be revealed by the server.

It is possible to use a tuple of local randomizers with edge LDP to obtain a relationship DP guarantee:

Proposition 1 (Edge LDP and relationship DP [29])**.** *If each of local randomizers $\mathcal{R}_1, \ldots, \mathcal{R}_n$ provides ε-edge LDP, then $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ provides 2ε-relationship DP. Additionally, if each $\mathcal{R}_i$ uses only bits $a_{i,1}, \ldots, a_{i,i-1}$ for users with smaller IDs (i.e., only the lower triangular part of $\mathbf{A}$), then $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ provides ε-relationship DP.*

The doubling factor in ε comes from the fact that (2) applies to an entire edge, whereas (1) applies to just one neighbor list, and adding an entire edge may cause changes to two neighbor lists. However, if each $\mathcal{R}_i$ ignores bits $a_{i,i}, \ldots, a_{i,n}$ for users with larger IDs, then this doubling factor can be avoided. Our algorithms also use only the lower triangular part of $\mathbf{A}$ to avoid this doubling issue.

Interaction among Users and Multiple Rounds. While interaction in LDP has been studied before [32], neither of Definitions 1 and 2 allows the interaction among users in a one-round protocol where user v_i sends $\mathcal{R}_i(\mathbf{a}_i)$ to the server.

However, the interaction among users is possible in a multi-round protocol. Specifically, at the first round, user v_i applies a randomizer $\mathcal{R}_i^1$ and sends $\mathcal{R}_i^1(\mathbf{a}_i)$ to the server. At the second round, the server calculates a message M_i for v_i by performing some post-processing on $\mathcal{R}_i^1(\mathbf{a}_i)$, possibly with the private outputs by other users. Let λ_i be the post-processing algorithm on $\mathcal{R}_i^1(\mathbf{a}_i)$; i.e., $M_i = \lambda_i(\mathcal{R}_i^1(\mathbf{a}_i))$. The server sends M_i to v_i. Then, v_i uses a randomizer $\mathcal{R}_i^2(M_i)$ that depends on M_i and sends $\mathcal{R}_i^2(M_i)(\mathbf{a}_i)$ back to the server. This entire computation provides DP by a (general) sequential composition [39]:

Proposition 2 (Sequential composition of edge LDP)**.** *For $i \in [n]$, let $\mathcal{R}_i^1$ be a local randomizer of user v_i that takes $\mathbf{a}_i$ as input. Let λ_i be a post-processing algorithm on $\mathcal{R}_i^1(\mathbf{a}_i)$, and $M_i = \lambda_i(\mathcal{R}_i^1(\mathbf{a}_i))$ be its output. Let $\mathcal{R}_i^2(M_i)$ be a local randomizer of v_i that depends on M_i. If $\mathcal{R}_i^1$ provides ε_1-edge LDP and for any $M_i \in \mathrm{Range}(\lambda_i)$, $\mathcal{R}_i^2(M_i)$ provides ε_2-edge LDP, then the sequential composition $(\mathcal{R}_i^1(\mathbf{a}_i), \mathcal{R}_i^2(M_i)(\mathbf{a}_i))$ provides $(\varepsilon_1 + \varepsilon_2)$-edge LDP.*

We provide a proof of Proposition 2 in the full version [30].

Global Sensitivity. We use the notion of global sensitivity [25] to provide edge LDP:

Definition 3. *In edge LDP (Definition 1), the global sensitivity of a function $f : \{0,1\}^n \to \mathbb{R}$ is given by:*

$$GS_f = \max_{\mathbf{a}_i, \mathbf{a}'_i \in \{0,1\}^n, \mathbf{a}_i \sim \mathbf{a}'_i} |f(\mathbf{a}_i) - f(\mathbf{a}'_i)|,$$

where $\mathbf{a}_i \sim \mathbf{a}'_i$ represents that $\mathbf{a}_i$ and $\mathbf{a}'_i$ differ in one bit.

For example, adding the Laplacian noise with mean 0 scale $\frac{GS_f}{\varepsilon}$ (denoted by $\mathrm{Lap}(\frac{GS_f}{\varepsilon})$) to $f(\mathbf{a}_i)$ provides ε-edge LDP.

3.3 Utility and Communication-Efficiency

Utility. We consider a private estimate of $f_\triangle(G)$. Our private estimator $\hat{f}_\triangle : \mathcal{G} \to \mathbb{R}$ is a post-processing of local randomizers $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ that satisfy ε-edge LDP. Following previous work, we use the l_2 loss (i.e., squared error) [33, 44, 60] and the relative error [16, 19, 63] as utility metrics.

Specifically, let l_2^2 be the expected l_2 loss function on a graph G, which maps the estimate $\hat{f}_\triangle(G)$ and the true value $f_\triangle(G)$ to the expected l_2 loss; i.e., $l_2^2(f_\triangle(G), \hat{f}_\triangle(G)) = \mathbb{E}[(\hat{f}_\triangle(G) - f_\triangle(G))^2]$. The expectation is taken over the randomness in the estimator $\hat{f}$, which is necessarily a randomized algorithm since it satisfies edge LDP. In our theoretical analysis, we analyze the expected l_2 loss, as with [33, 44, 60].

Note that the l_2 loss is large when $f_\triangle(G)$ is large. Therefore, in our experiments, we use the relative error given by $\frac{|\hat{f}_\triangle(G) - f_\triangle(G)|}{\max\{f_\triangle(G), \eta\}}$, where $\eta \in \mathbb{R}_{\geq 0}$ is a small value. Following convention [16, 19, 63], we set η to $0.001n$. The estimate is very accurate when the relative error is much smaller than 1.

Communication-Efficiency. A prominent concern when performing local computations is that the computing power of individual users is often limited. Of particular concern to our private estimators, and a bottleneck of previous work in locally private triangle counting [29], is the communication overhead between users and the server. This communication takes the form of users *downloading* any necessary data required to compute their local randomizers and *uploading* the output of their local randomizers. We distinguish the two quantities because often downloading is cheaper than uploading.

Consider a τ-round protocol, where $\tau \in \mathbb{N}$. At round $j \in [\tau]$, user v_i applies a local randomizer $\mathcal{R}_i^j(M_i^j)$ to her neighbor list $\mathbf{a}_i$, where M_i^j is a message sent from the server to user v_i during round j. We define the *download cost* as the number of bits required to describe M_i^j and the *upload cost* as the number of bits required to describe $\mathcal{R}_i^j(M_i^j)(\mathbf{a}_i)$. Over all rounds and all users, we evaluate the *maximum per-user download/upload cost*, which is given by:

$$\text{Cost}_{DL} = \max_{i=1}^{n} \sum_{j=1}^{\tau} \mathbb{E}[|M_i^j|] \quad \text{(bits)} \tag{3}$$

$$\text{Cost}_{UL} = \max_{i=1}^{n} \sum_{j=1}^{\tau} \mathbb{E}[|\mathcal{R}_i^j(M_i^j)(\mathbf{a}_i)|] \quad \text{(bits)}. \tag{4}$$

The above expectations go over the probability distributions of computing the local randomizers and any post-processing done by the server. We evaluate the maximum of the expected download/upload cost over users.

4 Communication-Efficient Triangle Counting Algorithms

The current state-of-the-art triangle counting algorithm [29] under edge LDP suffers from an extremely large per-user download cost; e.g., every user has to download a message of 400 Gbits or more when $n = 900000$. Therefore, it is impractical for a large graph. To address this issue, we propose three communication-efficient triangle algorithms under edge LDP.

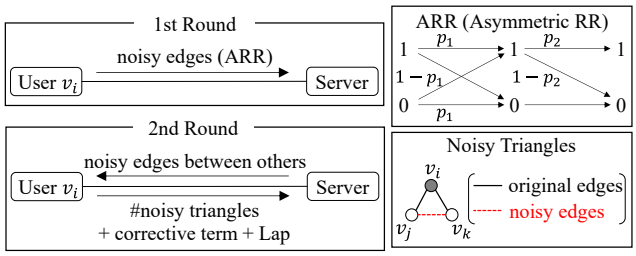


Figure 2: Overview of our communication-efficient triangle counting algorithms ($p_1 = \frac{e^\varepsilon}{e^\varepsilon+1}$, $p_2 \in [0,1]$).

We explain the overview and details of our proposed algorithms in Sections 4.1 and 4.2, respectively. Then we analyze the theoretical properties of our algorithms in Section 4.3.

4.1 Overview

Motivation. The drawback of the triangle algorithm in [29] is a prohibitively high download cost at the second round. This comes from the fact that in their algorithm, each user v_i applies Warner's RR (Randomized Response) [61] to bits for smaller user IDs in her neighbor list $\mathbf{a}_i$ (i.e., lower triangular part of $\mathbf{A}$) and then downloads the whole noisy graph. Since Warner's RR outputs 1 (edge) with high probability (e.g., about 0.5 when ε is close to 0), the number of edges in the noisy graph is extremely large—about half of the $\binom{n}{2}$ possible edges will be edges.

In this paper, we address this issue by introducing two strategies: *sampling edges* and *selecting edges each user downloads*. First, each user v_i samples each 1 (edge) after applying Warner's RR. Edge sampling has been widely studied in a non-private triangle counting problem [14, 26, 58, 62]. In particular, Wu *et al.* [62] compare various non-private triangle algorithms (e.g., edge sampling, node sampling, triangle sampling) and show that edge sampling provides almost the lowest estimation error. They also formally prove that edge sampling outperforms node sampling. Thus, sampling edges after Warner's RR is a natural choice for our private setting.

Second, we propose three strategies for selecting edges each user downloads. The first strategy is to simply select all noisy edges; i.e., each user downloads the whole noisy graph in the same way as [29]. The second and third strategies select some edges (rather than all edges) in a more clever manner so that the estimation error is significantly reduced. We provide a more detailed explanation in Section 4.2.

Algorithm Overview. Figure 2 shows the overview of our proposed algorithms.

At the first round, each user v_i obfuscates bits for smaller user IDs in her neighbor list $\mathbf{a}_i$ by an LDP mechanism which we call the *ARR (Asymmetric Randomized Response)* and

sends the obfuscated bits to a server. The ARR is a combination of Warner's RR and edge sampling; i.e., we apply Warner's RR that outputs 1 or 0 as it is with probability p_1 ($= \frac{e^\varepsilon}{e^\varepsilon+1}$) and then sample each 1 with probability $p_2 \in [0,1]$. Unlike Warner's RR, the ARR is asymmetric in that the flip probability in the whole process is different depending on the input value. As with Warner's RR, the ARR provides edge LDP. We can also significantly reduce the number of 1s (hence the communication cost) by setting p_2 small.

At the second round, the server calculates a message M_i for user v_i consisting of some or all noisy edges between others. We propose three strategies for calculating M_i. User v_i downloads M_i from the server. Then, since user v_i knows her edges, v_i can count *noisy triangles* (v_i, v_j, v_k) such that $j < k < i$ and only one edge (v_j, v_k) is noisy, as shown in Figure 2. The condition $j < k < i$ is imposed to use only the lower triangular part of $\mathbf{A}$, i.e., to avoid the doubling issue in Section 3.2. User v_i adds a corrective term and the Laplacian noise to the noisy triangle count and sends it to a server. The corrective term is added to enable the server to obtain an unbiased estimate of $f_\triangle(G)$. The Laplacian noise provides edge LDP. Finally, the server calculates an unbiased estimate of $f_\triangle(G)$ from the noisy data sent by users. By composition (Proposition 2), our algorithms provide edge LDP in total.

Remark. Note that it is also possible for the server to calculate an unbiased estimate of $f_\triangle(G)$ at the first round. However, this results in a prohibitively large estimation error because all edges sent by users are noisy; i.e., three edges are noisy in any triangle. In contrast, only one edge is noisy in each noisy triangle at the second round because each user v_i knows two original edges connected to v_i. Consequently, we can obtain an unbiased estimate with a much smaller variance. See Appendix B for a detailed comparison.

4.2 Algorithms

ARR. First, we formally define the ARR. The ARR has two parameters: $\varepsilon \in \mathbb{R}_{\geq 0}$ and $\mu \in [0, \frac{e^\varepsilon}{e^\varepsilon+1}]$. The parameter ε is the privacy budget, and μ controls the communication cost.

Let $ARR_{\varepsilon,\mu}$ be the ARR with parameters ε and μ. It takes $0/1$ as input and outputs $0/1$ with the following probability:

$$\Pr[ARR_{\varepsilon,\mu}(1) = b] = \begin{cases} \mu & (b=1) \\ 1-\mu & (b=0) \end{cases} \tag{5}$$

$$\Pr[ARR_{\varepsilon,\mu}(0) = b] = \begin{cases} \mu\rho & (b=1) \\ 1-\mu\rho & (b=0), \end{cases} \tag{6}$$

where $\rho = e^{-\varepsilon}$. By Figure 2, we can view this randomizer as a combination of Warner's RR [61] and edge sampling, where $\mu = p_1 p_2$. In fact, the ARR with $\mu = p_1 = \frac{e^\varepsilon}{e^\varepsilon+1}$ (i.e., $p_2 = 1$) is equivalent to Warner's RR.

Each user v_i applies the ARR to bits for smaller user IDs in her neighbor list $\mathbf{a}_i$; i.e., $\mathcal{R}_i(\mathbf{a}_i) = (ARR_{\varepsilon,\mu}(a_{i,1}), \ldots, ARR_{\varepsilon,\mu}(a_{i,i-1}))$. Then v_i sends $\mathcal{R}_i(\mathbf{a}_i)$ to the server. Since applying Warner's RR to $\mathbf{a}_i$ provides ε-edge LDP (as described in Section 3.2) and the sampling is a post-processing process, applying the ARR to $\mathbf{a}_i$ also provides ε-edge LDP by the immunity to post-processing [25].

Let $E' \subseteq V \times V$ be a set of noisy edges sent by users.

Which Noisy Edges to Download? Now, the main question tackled in this paper is: *Which noisy edges should each user v_i download at the second round?* Note that user v_i is not allowed to download only a set of noisy edges that form noisy triangles (i.e., $\{(v_j,v_k) \in E' | (v_i,v_j) \in E, (v_i,v_k) \in E\}$), because it tells the server who are friends with v_i. In other words, user v_i cannot leak her original edges to the server when she downloads noisy edges; the server must choose which part of E' to include in the message M_i it sends her.

Thus, a natural solution would be to download *all noisy edges between others* (with smaller user IDs); i.e., $M_i = \{(v_j,v_k) \in E' | j < k < i\}$. We denote our algorithm with this full download strategy by $\mathsf{ARRFull}_\triangle$. The (inefficient) two-rounds algorithm in [29] is a special case of $\mathsf{ARRFull}_\triangle$ without sampling ($\mu = p_1$). In other words, $\mathsf{ARRFull}_\triangle$ is a generalization of the two-rounds algorithm in [29] using the ARR.

In this paper, we show that we can do much better than $\mathsf{ARRFull}_\triangle$. Specifically, we prove in Section 4.3 that $\mathsf{ARRFull}_\triangle$ results in a high estimation error when the number of 4-cycles (cycles of length 4) in G is large. Intuitively, this can be explained as follows. Suppose that v_i, v_j, $v_{i'}$, and v_k ($j < k < i$, $j < k < i'$) form a 4-cycle. There is no triangle in this graph. However, if there is a noisy edge between v_j and v_k, then two (incorrect) noisy triangles appear: (v_i, v_j, v_k) counted by v_i and $(v_{i'}, v_j, v_k)$ counted by $v_{i'}$. More generally, let E_{ijk} (resp. $E_{i'jk}$) $\in \{0,1\}$ be a random variable that takes 1 if (v_i, v_j, v_k) (resp. $(v_{i'}, v_j, v_k)$) forms a noisy triangle and 0 otherwise. Then, the covariance $\mathrm{Cov}(E_{ijk}, E_{i'jk})$ between E_{ijk} and $E_{i'jk}$ is large because the presence/absence of a single noisy edge (v_j, v_k) affects the two noisy triangles.

To address this issue, we introduce a trick that makes the two noisy triangles *less correlated with each other*. We call this the *4-cycle trick*. Specifically, we propose two algorithms in which the server uses noisy edges connected to v_i when it calculates a message M_i for v_i. In the first algorithm, the server selects noisy edges (v_j, v_k) such that one noisy edge is connected from v_k to v_i; i.e., $M_i = \{(v_j,v_k) \in E' | (v_i,v_k) \in E', j < k < i\}$. We call this algorithm $\mathsf{ARROneNS}_\triangle$, as one noisy edge is connected to v_i. In the second algorithm, the server selects noisy edges (v_j, v_k) such that two noisy edges are connected from these nodes to v_i; i.e., $M_i = \{(v_j,v_k) \in E' | (v_i,v_j) \in E', (v_i,v_k) \in E', j < k < i\}$. We call this algorithm $\mathsf{ARRTwoNS}_\triangle$, as two noisy edges are connected to v_i. Note that user v_i does not leak her original edges to the server at the time of download in these algorithms, because the server uses only noisy edges E' sent by users to calculate M_i.

Figure 3 shows our three algorithms. The download cost Cost_{DL} in (3) is $O(\mu n^2 \log n)$, $O(\mu^2 n^2 \log n)$, and

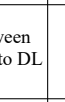
	ARRFull_Δ	ARROneNS_Δ	ARRTwoNS_Δ
Noisy edges between others (v_j and v_k) to DL			
Cost_{DL}	$O(\mu n^2 \log n)$	$O(\mu^2 n^2 \log n)$	$O(\mu^3 n^2 \log n)$

Figure 3: Noisy edges to download in our three algorithms.

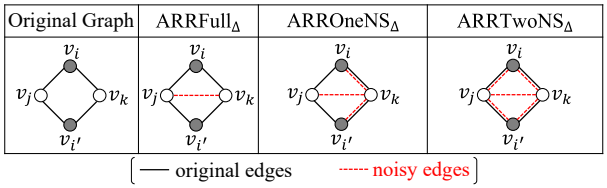


Figure 4: 4-cycle trick. $\text{ARRFull}_\triangle$ counts two (incorrect) noisy triangles when one noisy edge appears. $\text{ARROneNS}_\triangle$ and $\text{ARRTwoNS}_\triangle$ avoid this by increasing independent noise.

$O(\mu^3 n^2 \log n)$, respectively, when we regard ε as a constant. In our experiments, we set the parameter μ in the ARR so that μ in $\text{ARRFull}_\triangle$ is equal to μ^2 in $\text{ARROneNS}_\triangle$ and also equal to μ^3 in $\text{ARRTwoNS}_\triangle$; e.g., $\mu = 10^{-6}$, 10^{-3}, and 10^{-2} in $\text{ARRFull}_\triangle$, $\text{ARROneNS}_\triangle$, and $\text{ARRTwoNS}_\triangle$, respectively. Then the download cost is the same between the three algorithms.

Figure 4 shows our 4-cycle trick. $\text{ARRFull}_\triangle$ counts two (incorrect) noisy triangles when a noisy edge (v_j, v_k) appears. In contrast, $\text{ARROneNS}_\triangle$ (resp. $\text{ARRTwoNS}_\triangle$) counts both the two noisy triangles only when three (resp. five) independent noisy edges appear, as shown in Figure 4. Thus, this bad event happens with a much smaller probability. For example, $\text{ARRFull}_\triangle$ ($\mu = 10^{-6}$), $\text{ARROneNS}_\triangle$ ($\mu = 10^{-3}$), and $\text{ARRTwoNS}_\triangle$ ($\mu = 10^{-2}$) count both the two noisy triangles with probability 10^{-6}, 10^{-9}, and 10^{-10}, respectively. The covariance $\text{Cov}(E_{ijk}, E_{i'jk})$ of $\text{ARROneNS}_\triangle$ and $\text{ARRTwoNS}_\triangle$ is also much smaller than that of $\text{ARRFull}_\triangle$.

In our experiments, we show that $\text{ARROneNS}_\triangle$ and $\text{ARRTwoNS}_\triangle$ significantly outperforms $\text{ARRFull}_\triangle$ for a large-scale graph or dense graph, in both of which the number of 4-cycles in G is large.

$\text{ARROneNS}_\triangle$ vs. $\text{ARRTwoNS}_\triangle$. One might expect that $\text{ARRTwoNS}_\triangle$ outperforms $\text{ARROneNS}_\triangle$ because $\text{ARRTwoNS}_\triangle$ addresses the 4-cycle issue more aggressively; i.e., the number of independent noisy edges in a 4-cycle is larger in $\text{ARRTwoNS}_\triangle$, as shown in Figure 4. However, $\text{ARROneNS}_\triangle$ can reduce the global sensitivity of the Laplacian noise at the second round more effectively than $\text{ARRTwoNS}_\triangle$, as explained in Section 5. Consequently, $\text{ARROneNS}_\triangle$, which is the most tricky algorithm, achieves the smallest estimation error in our experiments. See Sections 5 and 6 for details of the global sensitivity and experiments, respectively.

Three Algorithms. Below we explain the details of our three algorithms. For ease of explanation, we assume that the maximum degree d_{max} is public in Section 4.2[2]. Note, however, that our double clipping (which is proposed to significantly reduce the global sensitivity) in Section 5 does *not* assume that d_{max} is public. Consequently, our entire algorithms do *not* require the assumption that d_{max} is public.

Input: Graph $G \in \mathcal{G}$ represented as neighbor lists $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \{0,1\}^n$, privacy budgets $\varepsilon_1, \varepsilon_2 \in \mathbb{R}_{\geq 0}$, $d_{max} \in \mathbb{Z}_{\geq 0}$, $\mu \in [0, \frac{e^{\varepsilon_1}}{e^{\varepsilon_1}+1}]$.
Output: Private estimate $\hat{f}_\triangle(G)$ of $f_\triangle(G)$.

1 [s] $\rho \leftarrow e^{-\varepsilon_1}$;
2 [v_i, s] $\mu^* \leftarrow \mu$, μ^2, and μ^3 in F, O, and T, respectively;
/* First round. */
3 **for** $i = 1$ **to** n **do**
4 [v_i] $\mathbf{r}_i \leftarrow (ARR_{\varepsilon_1,\mu}(a_{i,1}), \ldots, ARR_{\varepsilon_1,\mu}(a_{i,i-1}))$;
5 [v_i] Upload $\mathbf{r}_i = (r_{i,1}, \ldots, r_{i,i-1})$ to the server;
6 **end**
7 [s] $E' = \{(v_j, v_k) : r_{k,j} = 1, j < k\}$;
/* Second round. */
8 **for** $i = 1$ **to** n **do**
9 [s] Compute M_i by (7), (8), and (9) in F, O, and T, respectively;
10 [v_i] Download M_i from the server;
11 [v_i] $t_i \leftarrow |\{(v_i, v_j, v_k) : a_{i,j} = a_{i,k} = 1, (v_j, v_k) \in M_i, j < k < i\}|$;
12 [v_i] $s_i \leftarrow |\{(v_i, v_j, v_k) : a_{i,j} = a_{i,k} = 1, j < k < i\}|$;
13 [v_i] $w_i \leftarrow t_i - \mu^* \rho s_i$;
14 [v_i] $\hat{w}_i \leftarrow w_i + \text{Lap}(\frac{d_{max}}{\varepsilon_2})$;
15 [v_i] Upload $\hat{w}_i$ to the server;
16 **end**
17 [s] $\hat{f}_\triangle(G) \leftarrow \frac{1}{\mu^*(1-\rho)} \sum_{i=1}^n \hat{w}_i$;
18 **return** $\hat{f}_\triangle(G)$

Algorithm 1: Our three algorithms. "F", "O", "T" are shorthands for $\text{ARRFull}_\triangle$, $\text{ARROneNS}_\triangle$, and $\text{ARRTwoNS}_\triangle$, respectively. [$v_i$] and [s] represent that the process is run by v_i and the server, respectively.

Recall that the server calculates a message M_i for v_i as:

$$M_i = \{(v_j, v_k) \in E' | j < k < i\} \quad (7)$$
$$M_i = \{(v_j, v_k) \in E' | (v_i, v_k) \in E', j < k < i\} \quad (8)$$
$$M_i = \{(v_j, v_k) \in E' | (v_i, v_j) \in E', (v_i, v_k) \in E', j < k < i\} \quad (9)$$

in $\text{ARRFull}_\triangle$, $\text{ARROneNS}_\triangle$, $\text{ARRTwoNS}_\triangle$, respectively.

Algorithm 1 shows our three algorithms. These algorithms are processed differently in lines 2 and 9; "F", "O", "T" are shorthands for $\text{ARRFull}_\triangle$, $\text{ARROneNS}_\triangle$, and $\text{ARRTwoNS}_\triangle$,

[2]For example, d_{max} is public in Facebook: $d_{max} = 5000$ [3]. If the server does not have prior knowledge about d_{max}, she can privately estimate d_{max} and use graph projection to guarantee that each user's degree never exceeds the private estimate of d_{max} [29]. In any case, the assumption in Section 4.2 does not undermine our algorithms, because our entire algorithms with double clipping in Section 5 does *not* assume that d_{max} is public.

respectively. The privacy budgets for the first and second rounds are $\varepsilon_1, \varepsilon_2 \in \mathbb{R}_{\geq 0}$, respectively.

The first round appears in lines 3-7 of Algorithm 1. In this round, each user applies $ARR_{\varepsilon_1,\mu}$ defined by (5) and (6) to bits $a_{i,1}, \ldots, a_{i,i-1}$ for smaller user IDs in her neighbor list $\mathbf{a}_i$, i.e., lower triangular part of $\mathbf{A}$. Let $\mathbf{r}_i = (r_{i,1}, \ldots, r_{i,i-1}) \in \{0,1\}^{i-1}$ be the obfuscated bits of v_i. User v_i uploads $\mathbf{r}_i$ to the server. Then the server combines the noisy edges together, forming $E' = \{(v_j, v_k) : r_{k,j} = 1, j < k\}$.

The second round appears in lines 8-17 of Algorithm 1. In this round, the server computes a message M_i by (7), (8), or (9), and user v_i downloads it. Then user v_i calculates the number $t_i \in \mathbb{Z}_{\geq 0}$ of noisy triangles (v_i, v_j, v_k) such that only one edge (v_j, v_k) is noisy, as shown in Figure 2. User v_i also calculate a corrective term $s_i \in \mathbb{Z}_{\geq 0}$. The corrective term s_i is the number of possible triangles involving v_i and is computed to obtain an unbiased estimate of $f_\triangle(G)$. User v_i calculates $w_i = t_i - \mu^* \rho s_i$, where $\rho = e^{-\varepsilon_1}$ and $\mu^* = \mu, \mu^2$, and μ^3 in "F", "O", and "T", respectively. Then v_i adds the Laplacian noise $\mathrm{Lap}(\frac{d_{max}}{\varepsilon_2})$ to w_i to provide ε_2-edge LDP and sends the noisy value $\hat{w}_i$ $(= w_i + \mathrm{Lap}(\frac{d_{max}}{\varepsilon_2}))$ to the server. Note that adding one edge increases both t_i and s_i by at most d_{max}. Thus, the global sensitivity of w_i is at most d_{max}. Finally, the server calculates an estimate of $f_\triangle(G)$ as: $\hat{f}_\triangle(G) = \frac{1}{\mu^*(1-\rho)} \sum_{i=1}^{n} \hat{w}_i$. As we prove later, $\hat{f}_\triangle(G)$ is an unbiased estimate of $f_\triangle(G)$.

4.3 Theoretical Analysis

We now introduce the theoretical guarantees on the privacy, communication, and utility of our algorithms.

Privacy. We first show the privacy guarantees:

Theorem 1. *For $i \in [n]$, let $\mathcal{R}_i^1, \mathcal{R}_i^2(M_i)$ be the randomizers used by user v_i in rounds 1 and 2 of Algorithm 1. Let $\mathcal{R}_i(\mathbf{a}_i) = (\mathcal{R}_i^1(\mathbf{a}_i), \mathcal{R}_i^2(M_i)(\mathbf{a}_i))$ be the composition of the two randomizers. Then, $\mathcal{R}_i$ satisfies $(\varepsilon_1 + \varepsilon_2)$-edge LDP and $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ satisfies $(\varepsilon_1 + \varepsilon_2)$-relationship DP.*

Note that the doubling issue in Section 3.2 does not occur, because we use only the lower triangular part of $\mathbf{A}$. By the immunity to post-processing, the estimate $\hat{f}_\triangle(G)$ also satisfies $(\varepsilon_1 + \varepsilon_2)$-edge LDP and $(\varepsilon_1 + \varepsilon_2)$-relationship DP.

Communication. Recall that we evaluate the algorithms based on their download cost (3) and upload cost (4).

Download Cost: The download cost is the number of bits required to download M_i. M_i can be represented as a list of edges between others, and each edge can be identified with two indices (user IDs), i.e., $2\log n$ bits. There are $\frac{(n-1)(n-2)}{2} \approx \frac{n^2}{2}$ edges between others. $ARR_{\varepsilon_1,\mu}$ outputs 1 with probability at most μ. In addition, each noisy triangle must have 1, 2, and 3 noisy edges in ARRFull$_\triangle$, ARROneNS$_\triangle$, and ARRTwoNS$_\triangle$, respectively, as shown in Figure 3.

Thus, the download cost in Algorithm 1 can be written as:

$$Cost_{DL} \leq \mu^* n^2 \log n, \tag{10}$$

where $\mu^* = \mu, \mu^2$, and μ^3 in ARRFull$_\triangle$, ARROneNS$_\triangle$, and ARRTwoNS$_\triangle$, respectively. In (10), we upper-bounded Cost_{DL} by using the fact that $ARR_{\varepsilon_1,\mu}$ outputs 1 with probability at most μ. However, when $d_{max} \ll n$, $ARR_{\varepsilon_1,\mu}$ outputs 1 with probability $\mu e^{-\varepsilon_1}$ in most cases. In that case, we can roughly approximate Cost_{DL} by replacing μ with $\mu e^{-\varepsilon_1}$ in (10).

Upload Cost: The upload cost comes from the number of bits required to upload $\mathcal{R}_i^1(\mathbf{a}_i)$ and $\mathcal{R}_i^2(M_i)(\mathbf{a}_i)$. Uploading $\mathcal{R}_i^1(\mathbf{a}_i)$ involves uploading $\mathbf{r}_i$ (line 5), which is a list of up to n noisy neighbors. By sending just the indices (user IDs) of the 1s in $\mathbf{r}_i$, each user sends $\|\mathbf{r}_i\|_1 \log n$ bits, where $\|\mathbf{r}_i\|_1$ is the number of 1s in $\mathbf{r}_i$. When we use $ARR_{\varepsilon_1,\mu}$, we have $\mathbb{E}[\|\mathbf{r}_i\|_1] \leq \mu n$. Uploading $\mathcal{R}_i^2(M_i)$ involves uploading a single real number $\hat{w}_i$ (line 15), which is negligibly small (e.g., 64 bits when we use a double-precision floating-point).

Thus, the upload cost in Algorithm 1 can be written as:

$$Cost_{UL} \leq \mu n \log n. \tag{11}$$

Clearly, Cost_{UL} is much smaller than Cost_{DL} for large n.

Utility. Analyzing the expected l_2 loss $l_2^2(f_\triangle(G), \hat{f}_\triangle(G))$ of the algorithms involves first proving that the estimator $\hat{f}_\triangle$ is unbiased and then analyzing the variance $\mathbb{V}[\hat{f}_\triangle(G)]$ to obtain an upper-bound on $l_2^2(f_\triangle(G), \hat{f}_\triangle(G))$. This is given in the following:

Theorem 2. *Let $G \in \mathcal{G}$, $\varepsilon_1, \varepsilon_2 \in \mathbb{R}_{\geq 0}$, and $\mu \in [0, \frac{e^{\varepsilon_1}}{e^{\varepsilon_1}+1}]$. Let $\hat{f}_\triangle^F(G), \hat{f}_\triangle^O(G)$, and $\hat{f}_\triangle^T(G)$ be the estimates output respectively by ARRFull$_\triangle$, ARROneNS$_\triangle$, and ARRTwoNS$_\triangle$ in Algorithm 1. Then, $\mathbb{E}[\hat{f}_\triangle^F(G)] = \mathbb{E}[\hat{f}_\triangle^O(G)] = \mathbb{E}[\hat{f}_\triangle^T(G)] = f_\triangle(G)$ (i.e., estimates are unbiased) and*

$$l_2^2(f_\triangle(G), \hat{f}_\triangle^F(G)) \leq \frac{2C_4(G)+S_2(G)}{\mu(1-e^{\varepsilon_1})^2} + \frac{2nd_{max}^2}{\mu^2(1-e^{\varepsilon_1})^2\varepsilon_2^2}$$

$$l_2^2(f_\triangle(G), \hat{f}_\triangle^O(G)) \leq \frac{\mu(2C_4(G)+6S_3(G))+S_2(G)}{\mu^2(1-e^{\varepsilon_1})^2} + \frac{2nd_{max}^2}{\mu^4(1-e^{\varepsilon_1})^2\varepsilon_2^2}$$

$$l_2^2(f_\triangle(G), \hat{f}_\triangle^T(G)) \leq \frac{\mu^2(2C_4(G)+6S_3(G))+S_2(G)}{\mu^3(1-e^{\varepsilon_1})^2} + \frac{2nd_{max}^2}{\mu^6(1-e^{\varepsilon_1})^2\varepsilon_2^2},$$

where $C_4(G)$ is the number of 4-cycles in G and $S_k(G)$ is the number of k-stars in G.

For each of the three upper-bounds in Theorem 2, the first and second terms are the estimation errors caused by empirical estimation and the Laplacian noise, respectively. We also note that $C_4(G) = S_3(G) = O(nd_{max}^3)$ and $S_2(G) = O(nd_{max}^2)$. Thus, for small μ, the l_2 loss of empirical estimation can be expressed as $O(nd_{max}^3)$, $O(nd_{max}^2)$, and $O(nd_{max}^2)$ in ARRFull$_\triangle$, ARROneNS$_\triangle$, ARRTwoNS$_\triangle$, respectively (as the factors of $C_4(G)$ and $S_3(G)$ diminish for small μ).

This highlights our 4-cycle trick. The large l_2 loss of ARRFull$_\triangle$ is caused by the number $C_4(G) = O(nd_{max}^3)$ of 4-cycles. ARROneNS$_\triangle$ and ARRTwoNS$_\triangle$ addresses this issue by increasing independent noise, as shown in Figure 4.

5 Double Clipping

In Section 4, we showed that the estimation error caused by empirical estimation (i.e., the first term in Theorem 2) is significantly reduced by the 4-cycle trick. However, the estimation error is still very large in our algorithms presented in Section 4, as shown in our experiments. This is because the estimation error by the Laplacian noise (i.e., the second term in Theorem 2) is very large, especially for small ε_2 or μ. This error term is tight and unavoidable as long as we use d_{max} as a global sensitivity, which suggests that we need a better global sensitivity analysis. To significantly reduce the global sensitivity, we propose a novel *double clipping* technique.

We describe the overview and details of our double clipping in Sections 5.1 and 5.2, respectively. Then we perform theoretical analysis in Section 5.3.

5.1 Overview

Motivation. Figure 5 shows noisy triangles involving edge (v_i, v_j) counted by user v_i in our three algorithms. Our algorithms in Section 4 use the fact that the number of such noisy triangles (hence the global sensitivity) is upper-bounded by the maximum degree d_{max} because adding one edge increases the triangle count by at most d_{max}. Unfortunately, this upper-bound is too large, as shown in our experiments.

In this paper, we significantly reduce this upper-bound by using the parameter μ in the ARR and user v_i's degree $d_i \in \mathbb{Z}_{\geq 0}$ for users with smaller IDs. For example, the number of noisy triangles involving (v_i, v_j) in ARRFull$_\triangle$ is expected to be around μd_i because one noisy edge is included in each noisy triangle (as shown in Figure 5) and all noisy edges are independent. μd_i is very small, especially when we set $\mu \ll 1$ to reduce the communication cost.

However, we cannot directly use μd_i as an upper-bound of the global sensitivity in ARRFull$_\triangle$ for two reasons. First, μd_i leaks the exact value of user v_i's degree d_i and violates edge LDP. Second, the number of noisy triangles involving (v_i, v_j) exceeds μd_i with high probability (about 0.5). Thus, the noisy triangle count cannot be upper-bounded by μd_i.

To address these two issues, we propose a double clipping technique, which is explained below.

Algorithm Overview. Figure 6 shows the overview of our double clipping, which consists of an *edge clipping* and *noisy triangle clipping*. The edge clipping addresses the first issue (i.e., leakage of d_i) as follows. It privately computes a noisy version of d_i (denoted by $\tilde{d}_i$) with edge LDP. Then it removes some neighbors from a neighbor list $\mathbf{a}_i$ so that the degree of v_i never exceeds the noisy degree $\tilde{d}_i$. This removal process is also known as graph projection [23, 24, 37, 50]. Edge clipping is used in [29] to obtain a noisy version of d_{max}.

The main novelty in our double clipping lies at the *noisy triangle clipping* to address the second issue (i.e., excess of the noisy triangle count). This issue appears when we attempt

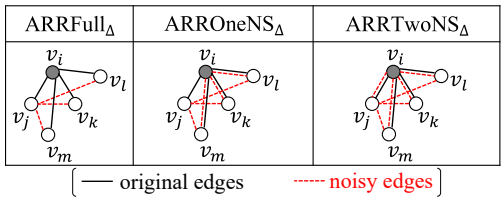


Figure 5: Noisy triangles involving edge (v_i, v_j) counted by user v_i ($j < k, l, m < i$).

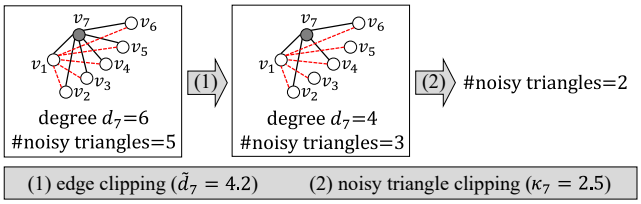


Figure 6: Overview of double clipping applied to edge (v_1, v_7).

to reduce the global sensitivity by using a very small sampling probability for each edge. Therefore, the noisy triangle clipping has not been studied in the existing works on private triangle counting [24, 29, 36, 37, 56, 64, 65, 67], because they do not apply a sampling technique.

Our noisy triangle clipping reduces the noisy triangle count so that it never exceeds a user-dependent clipping threshold $\kappa_i \in \mathbb{R}_{\geq 0}$. Then a crucial issue is how to set an appropriate threshold κ_i. We theoretically analyze the probability that the noisy triangle count exceeds κ_i (referred to as the *triangle excess probability*) as a function of the ARR parameter μ and the noisy degree $\tilde{d}_i$. Then we set κ_i so that the triangle excess probability is very small ($= 10^{-6}$ in our experiments).

We use the clipping threshold κ_i as a global sensitivity. Note that κ_i provides edge LDP because $\tilde{d}_i$ provides edge LDP, i.e., immunity to post-processing [25]. κ_i is also very small when $\mu \ll 1$, as it is determined based on μ.

5.2 Algorithms

Algorithm 2 shows our double clipping algorithm. All the processes are run by user v_i at the second round. Thus, there is no interaction with the server in Algorithm 2.

Edge Clipping. The edge clipping appears in lines 2-3 of Algorithm 2. It uses a privacy budget $\varepsilon_0 \in \mathbb{R}_{\geq 0}$.

In line 2, user v_i adds the Laplacian noise $\mathrm{Lap}(\frac{1}{\varepsilon_0})$ to her degree d_i. Since adding/removing one edge changes d_i by at most 1, this process provides ε_0-edge LDP. v_i also adds some non-negative constant $\alpha \in \mathbb{R}_{\geq 0}$ to d_i. We add this value so that edge removal (in line 3) occurs with a very small probability; e.g., in our experiments, we set $\alpha = 150$, where edge removal occurs with probability 1.5×10^{-7} when $\varepsilon_0 = 0.1$. A similar technique is introduced in [56] to provide (ε, δ)-DP [25] with small δ. The difference between ours and [56] is that we perform edge clipping to always provide ε-DP; i.e., $\delta = 0$. Let $\tilde{d}_i \in \mathbb{R}_{\geq 0}$ be the noisy degree of v_i.

Input: Neighbor list $\mathbf{a}_i \in \{0,1\}^n$, privacy budget $\varepsilon_0 \in \mathbb{R}_{\geq 0}$ $\mu \in [0, \frac{e^{\varepsilon_1}}{e^{\varepsilon_1}+1}]$, $\alpha \in \mathbb{R}_{\geq 0}$, $\beta \in \mathbb{R}_{\geq 0}$.
Output: $\hat{w}_i$.

1 $\mu^* \leftarrow \mu$, μ^2, and μ^3 in F, O, and T, respectively;
/* Edge clipping. */
2 $\tilde{d}_i = \max\{d_i + \mathrm{Lap}(\frac{1}{\varepsilon_0}) + \alpha, 0\}$;
/* Remove $d_i - \lfloor \tilde{d}_i \rfloor$ neighbors if $d_i > \tilde{d}_i$. */
3 $\mathbf{a}_i \leftarrow \mathtt{GraphProjection}(\mathbf{a}_i, \tilde{d}_i)$;
/* Noisy triangle clipping. */
4 **for** j such that $a_{i,j} = 1$ and $j < i$ **do**
5 $\quad t_{i,j} \leftarrow |\{(v_i, v_j, v_k) : a_{i,k} = 1, (v_j, v_k) \in M_i, j < k < i\}|$;
6 **end**
/* Calculate $\kappa_i \in [\mu^* \tilde{d}_i, \tilde{d}_i]$ s.t. the triangle excess probability is β or less. */
7 $\kappa_i \leftarrow \mathtt{ClippingThreshold}(\mu, \tilde{d}_i, \beta)$;
8 $t_i \leftarrow \sum_{a_{i,j}=1, j<i} \min\{t_{i,j}, \kappa_i\}$;
9 $s_i \leftarrow |\{(v_i, v_j, v_k) : a_{i,j} = a_{i,k} = 1, j < k < i\}|$;
10 $w_i \leftarrow t_i - \mu^* \rho s_i$;
11 $\hat{w}_i \leftarrow w_i + \mathrm{Lap}(\frac{\kappa_i}{\varepsilon_2})$;
12 **return** $\hat{w}_i$

Algorithm 2: Our double clipping algorithm. "F", "O", "T" are shorthands for ARRFull$_\triangle$, ARROneNS$_\triangle$, and ARRTwoNS$_\triangle$, respectively. All the processes are run by user v_i.

In line 3, user v_i calls the function `GraphProjection`, which performs graph projection as follows; if $d_i > \tilde{d}_i$, randomly remove $d_i - \lfloor \tilde{d}_i \rfloor$ neighbors from $\mathbf{a}_i$; otherwise, do nothing. Consequently, the degree of v_i never exceeds $\tilde{d}_i$.

Noisy Triangle Clipping. The noisy triangle clipping appears in lines 4-11 of Algorithm 2.

In lines 4-6, user v_i calculates the number $t_{i,j} \in \mathbb{Z}_{\geq 0}$ of noisy triangles (v_i, v_j, v_k) $(j < k < i)$ involving (v_i, v_j) (as shown in Figure 5). Note that the total number t_i of noisy triangles of v_i can be expressed as: $t_i = \sum_{a_{i,j}=1, j<i} t_{i,j}$. In line 7, v_i calls the function `ClippingThreshold`, which calculates a clipping threshold $\kappa_i \in [\mu^* \tilde{d}_i, \tilde{d}_i]$ ($\mu^* = \mu$, μ^2, and μ^3 in "F", "O", and "T", respectively) based on the ARR parameter μ and the noisy degree $\tilde{d}_i$ so that the triangle excess probability does not exceed some constant $\beta \in \mathbb{R}_{\geq 0}$. We explain how to calculate the triangle excess probability in Section 5.3. In line 8, v_i calculates the total number t_i of noisy triangles by summing up $t_{i,j}$, with the exception that v_i adds κ_i if $t_{i,j} > \kappa_i$. In other words, triangle removal occurs if $t_{i,j} > \kappa_i$. Then, the number of noisy triangles involving (v_i, v_j) never exceeds κ_i.

Lines 9-11 in Algorithm 2 are the same as lines 12-14 in Algorithm 1, except that the global sensitivity in the former (resp. latter) is κ_i (resp. d_{max}). Line 11 in Algorithm 2 provides ε_2-edge LDP because the number of triangles involving (v_i, v_j) is now upper-bounded by κ_i.

Our Entire Algorithms with Double Clipping. We can run our algorithms ARRFull$_\triangle$, ARROneNS$_\triangle$, ARRTwoNS$_\triangle$ with double clipping just by replacing lines 11-14 in Algorithm 1 with lines 2-11 in Algorithm 2. That is, after calculating $\hat{w}_i$ by Algorithm 2, v_i uploads $\hat{w}_i$ to the server. Then the server calculates an estimate of $f_\triangle(G)$ as $\hat{f}_\triangle(G) = \frac{1}{\mu^*(1-\rho)} \sum_{i=1}^n \hat{w}_i$.

We also note that the input d_{max} in Algorithm 1 is no longer necessary thanks to the edge clipping; i.e., our entire algorithms with double clipping do not assume that d_{max} is public.

5.3 Theoretical Analysis

We now perform a theoretical analysis on the privacy and utility of our double clipping.

Privacy. We begin with the privacy guarantees:

Theorem 3. *For $i \in [n]$, let $\mathcal{R}_i^1, \mathcal{R}_i^2(M_i)$ be the randomizers used by user v_i in rounds 1 and 2 of our algorithms with double clipping (Algorithms 1 and 2). Let $\mathcal{R}_i(\mathbf{a}_i) = (\mathcal{R}_i^1(\mathbf{a}_i), \mathcal{R}_i^2(M_i)(\mathbf{a}_i))$ be the composition of the two randomizers. Then, $\mathcal{R}_i$ satisfies $(\varepsilon_0 + \varepsilon_1 + \varepsilon_2)$-edge LDP, and $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ satisfies $(\varepsilon_0 + \varepsilon_1 + \varepsilon_2)$-relationship DP.*

Utility. Next, we show the triangle excess probability:

Theorem 4. *In Algorithm 2, the triangle excess probability (i.e., probability that the number of noisy triangles $t_{i,j}$ involving edge (v_i, v_j) exceeds a clipping threshold κ_i) is:*

$$\Pr(t_{i,j} > \kappa_i) \leq \exp\left[-\tilde{d}_i D\left(\frac{\kappa_i}{\tilde{d}_i} \parallel \mu\right)\right] \tag{12}$$

$$\Pr(t_{i,j} > \kappa_i) \leq \exp\left[-\tilde{d}_i D\left(\frac{\kappa_i}{\tilde{d}_i} \parallel \mu^2\right)\right] \tag{13}$$

$$\Pr(t_{i,j} > \kappa_i) \leq \mu \exp\left[-\tilde{d}_i D\left(\frac{\max\{\kappa_i, \mu^2 \tilde{d}_i\}}{\tilde{d}_i} \parallel \mu^2\right)\right] \tag{14}$$

in ARRFull$_\triangle$, ARROneNS$_\triangle$, and ARRTwoNS$_\triangle$, respectively, where $D(p_1 \parallel p_2)$ is the Kullback-Leibler divergence between two Bernoulli distributions; i.e.,

$$D(p_1 \parallel p_2) = p_1 \log \frac{p_1}{p_2} + (1 - p_1) \log \frac{1-p_1}{1-p_2}.$$

In all of (12), (13), and (14), we use the Chernoff bound, which is known to be reasonably tight [10].

Setting κ_i. The function `ClippingThreshold` in Algorithm 2 sets a clipping threshold κ_i of user v_i based on Theorem 4. Specifically, we set $\kappa_i = \lambda_i \mu^* \tilde{d}_i$, where $\lambda_i \in \mathbb{N}$, and calculate λ_i as follows. We initially set $\lambda_i = 1$ and keep increasing λ_i by 1 until the upper-bound (i.e., right-hand side of (12), (13), or (14)) is smaller than or equal to the triangle excess probability β. In our experiments, we set $\beta = 10^{-6}$.

Large κ_i of ARRTwoNS$_\triangle$. By (12) and (13), the upper-bound on the triangle excess probability is the same between ARRFull$_\triangle$ and ARROneNS$_\triangle$. In contrast, ARRTwoNS$_\triangle$ has a larger upper-bound. For example, when $\kappa_i = 15\mu^* \tilde{d}_i$, $\mu^* = 10^{-3}$, and $\tilde{d}_i = 1000$, the right-hand sides of (12), (13),

	ARRFull$_\triangle$	ARROneNS$_\triangle$	ARRTwoNS$_\triangle$
Privacy	$(\varepsilon_0+\varepsilon_1+\varepsilon_2)$-edge LDP and $(\varepsilon_0+\varepsilon_1+\varepsilon_2)$-relationship DP		
Expected l_2 loss	$O\left(\frac{nd_{max}^3}{\mu(1-e^{-\varepsilon_1})^2}+\frac{2\sum_{i=1}^n \kappa_i^2}{\mu^2(1-e^{-\varepsilon_1})^2\varepsilon_2^2}\right)$	$O\left(\frac{nd_{max}^2}{\mu^2(1-e^{-\varepsilon_1})^2}+\frac{2\sum_{i=1}^n \kappa_i^2}{\mu^4(1-e^{-\varepsilon_1})^2\varepsilon_2^2}\right)$	$O\left(\frac{nd_{max}^2}{\mu^3(1-e^{-\varepsilon_1})^2}+\frac{2\sum_{i=1}^n \kappa_i^2}{\mu^6(1-e^{-\varepsilon_1})^2\varepsilon_2^2}\right)$
Cost$_{DL}$	$\mu n^2 \log n$	$\mu^2 n^2 \log n$	$\mu^3 n^2 \log n$
Cost$_{UL}$	$\mu n \log n$	$\mu n \log n$	$\mu n \log n$

Table 1: Performance guarantees of our three algorithms with double clipping when the edge removal and triangle removal do not occur. The expected l_2 loss assumes that μ is small. The download (resp. upload) cost is an upper-bound in (10) (resp. (11)).

and (14) are 2.5×10^{-12}, 2.5×10^{-12}, and 3.3×10^{-2}, respectively. Consequently, ARRTwoNS$_\triangle$ has a larger global sensitivity κ_i for the same value of β.

We can explain a large global sensitivity κ_i of ARRTwoNS$_\triangle$ as follows. The number $t_{i,j}$ of noisy triangles involving (v_i, v_j) in ARRFull$_\triangle$ is expected to be around μd_i because one noisy edge is in each noisy triangle (as in Figure 5) and all noisy edges are independent. For the same reason, $t_{i,j}$ in ARROneNS$_\triangle$ is expected to be around $\mu^2 d_i$. However, $t_{i,j}$ in ARRTwoNS$_\triangle$ is *not* expected to be around $\mu^3 d_i$, because all the noisy triangles have noisy edge (v_i, v_j) in common (as in Figure 5). Then, the expectation of $t_{i,j}$ largely depends on the presence/absence of the noisy edge (v_i, v_j); i.e., if noisy edge (v_i, v_j) exists, it is $\mu^2 d_i$; otherwise, 0. Thus, κ_i cannot be effectively reduced by double clipping.

Summary. The performance guarantees of our three algorithms with double clipping can be summarized in Table 1.

The first and second terms of the expected l_2 loss are the l_2 loss of empirical estimation and that of the Laplacian noise, respectively. For small μ, the l_2 loss of empirical estimation can be expressed as $O(nd_{max}^3)$, $O(nd_{max}^2)$, and $O(nd_{max}^2)$ in ARRFull$_\triangle$, ARROneNS$_\triangle$, ARRTwoNS$_\triangle$, respectively, as explained in Section 4.3. The l_2 loss of the Laplacian noise is $O(\sum_{i=1}^n \kappa_i^2)$, which is much smaller than $O(nd_{max}^2)$. Thus, our ARROneNS$_\triangle$ that effectively reduces κ_i provides the smallest error, as shown in our experiments.

We also note that both the space and the time complexity to compute and send M_i in our algorithms are $O(\mu^* n^2)$ (as $|E'| = O(\mu^* n^2)$), which is much smaller than [29] ($= O(n^2)$).

6 Experiments

To evaluate each component of our algorithms in Sections 4 and 5 as well as our entire algorithms (i.e., ARRFull$_\triangle$, ARROneNS$_\triangle$, ARRTwoNS$_\triangle$with double clipping), we pose the following three research questions:

RQ1. How do our three triangle counting algorithms (i.e., ARRFull$_\triangle$, ARROneNS$_\triangle$, ARRTwoNS$_\triangle$) in Section 4 compare with each other in terms of accuracy?

RQ2. How much does our double clipping technique in Section 5 decrease the estimation error?

RQ3. How much do our entire algorithms reduce the communication cost, compared to the existing algorithm [29], while keeping high utility (e.g., relative error $\ll 1$)?

In Appendix B, we also compare our entire algorithms with one-round algorithms.

6.1 Experimental Set-up

In our experiments, we used two real graph datasets:

Gplus. The Google+ dataset [41] (denoted by Gplus) was collected from users who had shared circles. From the dataset, we constructed a social graph $G = (V, E)$ with 107614 nodes (users) and 12238285 edges, where edge $(v_i, v_j) \in E$ represents that v_i follows or is followed by v_j. The average (resp. maximum) degree in G is 113.7 (resp. 20127).

IMDB. The IMDB (Internet Movie Database) [2] (denoted by IMDB) includes a bipartite graph between 896308 actors and 428440 movies. From this, we constructed a graph $G = (V, E)$ with 896308 nodes (actors) and 57064358 edges, where edge $(v_i, v_j) \in E$ represents that v_i and v_j have played in the same movie. The average (resp. maximum) degree in G is 63.7 (resp. 15451). Thus, IMDB is more sparse than Gplus.

In the full version [30], we also evaluate our algorithms using a synthetic graph based on the Barabási-Albert model [11], which has a power-law degree distribution.

We evaluated our algorithms while changing μ^*, where $\mu^* = \mu$, μ^2, and μ^3 in ARRFull$_\triangle$, ARROneNS$_\triangle$, and ARRTwoNS$_\triangle$, respectively. Cost$_{DL}$ is the same between the three algorithms. We typically set the total privacy budget ε to $\varepsilon = 1$ (at most 2) because it is acceptable in many practical scenarios [39].

In our double clipping, we set $\alpha = 150$ and $\beta = 10^{-6}$ so that both edge removal and triangle removal occur with a very small probability ($\leq 10^{-6}$ when $\varepsilon_0 = 0.1$). Then for each algorithm, we evaluated the relative error between the true triangle count $f_\triangle(G)$ and its estimate $\hat{f}_\triangle(G)$. Since the estimate $\hat{f}_\triangle(G)$ varies depending on the randomness of LDP mechanisms, we ran each algorithm $\tau \in \mathbb{N}$ times ($\tau = 20$ and 10 for Gplus and IMDB, respectively) and averaged the relative error over the τ cases.

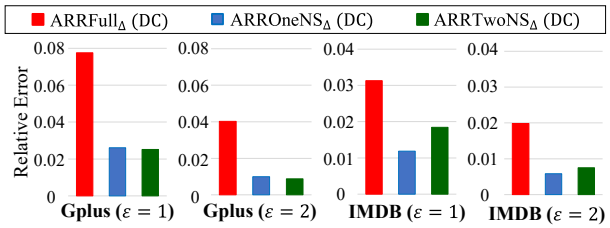


Figure 7: Relative error of our three algorithms with double clipping ("DC") when $\varepsilon = 1$ or 2 and $\mu^* = 10^{-3}$ ($n = 107614$ in Gplus, $n = 896308$ in IMDB).

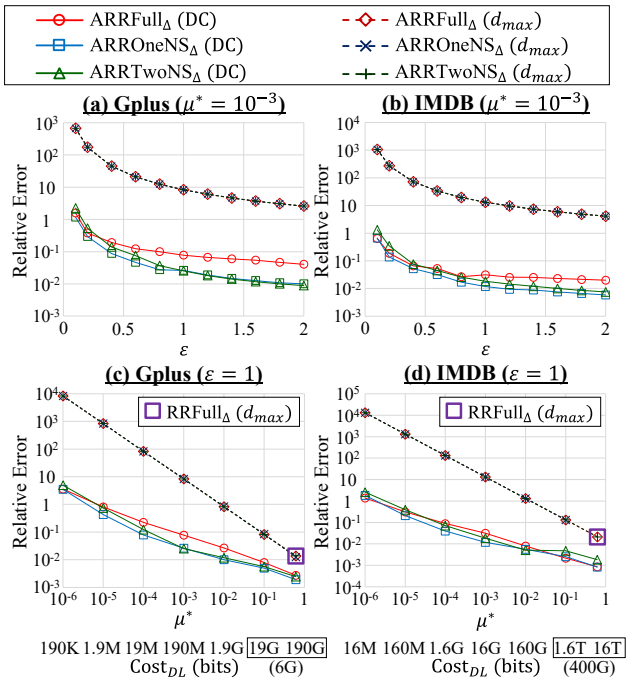


Figure 8: Relative error of our three algorithms with ("DC") or without ("d_{max}") double clipping ($n = 107614$ in Gplus, $n = 896308$ in IMDB). RRFull$_\Delta(d_{max})$ is the algorithm in [29]. Cost$_{DL}$ is an upper-bound in (10). When $\mu^* \geq 0.1$, Cost$_{DL}$ can be 6 Gbits and 400 Gbits in Gplus and IMDB, respectively, by downloading only 0/1 for each pair of users (v_j, v_k).

6.2 Experimental Results

Performance Comparison. First, we evaluated our algorithms with the Laplacian noise. Specifically, we evaluated all possible combinations of our three algorithms with and without our double clipping (six combinations in total) and compared them with the existing two-rounds algorithm in [29]. For algorithms with double clipping, we divided the total privacy budget ε as: $\varepsilon_0 = \frac{\varepsilon}{10}$ and $\varepsilon_1 = \varepsilon_2 = \frac{9\varepsilon}{20}$. Here, we set a very small budget ($\varepsilon_0 = \frac{\varepsilon}{10}$) for edge clipping because the degree has a small sensitivity (sensitivity= 1). For algorithms without double clipping, we divided ε as $\varepsilon_1 = \varepsilon_2 = \frac{\varepsilon}{2}$ and used the maximum degree d_{max} as the global sensitivity.

Figures 7 and 8 show the results. Figure 7 highlights the relative error of our three algorithms with double clipping when $\varepsilon = 1$ or 2 and $\mu^* = 10^{-3}$. "DC" (resp. "d_{max}") represents algorithms with (resp. without) double clipping. RRFull$_\Delta(d_{max})$ (marked with purple square) in Figure 8 (c) and (d) represents the two-rounds algorithm in [29]. Note that this is a special case of our ARRFull$_\Delta$ without sampling ($\mu = \frac{e^{\varepsilon_1}}{e^{\varepsilon_1}+1} = 0.62$). Figure 8 (c) and (d) also show the download cost Cost$_{DL}$ calculated by (10). Note that when $\mu^* \geq 0.1$ (marked with squares), Cost$_{DL}$ can be 6Gbits and 400Gbits in Gplus and IMDB, respectively, by downloading only 0/1 for each pair of users (v_j, v_k); Cost$_{DL} = \frac{(n-1)(n-2)}{2}$ in this case.

Figures 7 and 8 show that our ARROneNS$_\Delta$ (DC) provides the best (or almost the best) performance in all cases. This is because ARROneNS$_\Delta$ (DC) introduces the 4-cycle trick and effectively reduces the global sensitivity of the Laplacian noise by double clipping. Later, we will investigate the effectiveness of the 4-cycle trick in detail by not adding the Laplacian noise. We will also investigate the impact of the Laplacian noise while changing n.

Figure 8 also shows that the relative error is almost the same between our three algorithms without double clipping ("d_{max}") and that it is too large. This is because Lap($\frac{d_{max}}{\varepsilon_2}$) is too large and dominant. The relative error is significantly reduced by introducing our double clipping in all cases. For example, when $\mu^* = 10^{-3}$, our double clipping reduces the relative error of ARROneNS$_\Delta$ by two or three orders of magnitude. The improvement is larger for smaller μ^*.

In the full version [30], we also evaluate the effect of edge clipping and noisy triangle clipping independently and show that each component significantly reduces the relative error.

Communication Cost. From Figure 8 (c) and (d), we can explain how much our algorithms can reduce the download cost while keeping high utility, e.g., relative error $\ll 1$.

For example, when we use the algorithm in [29], the download cost is Cost$_{DL} = 400$ Gbits in IMDB. Thus, when the download speed is 20 Mbps (recommended speed in YouTube [7]), every user v_i needs 6 hours to download the message M_i, which is far from practical. In contrast, our ARROneNS$_\Delta$ (DC) can reduce it to 160 Mbits (8 seconds when 20 Mbps download rate) or less while keeping relative error $= 0.21$, which is practical and a dramatic improvement over [29].

We also note that since $d_{max} \ll n$ in IMDB, Cost$_{DL}$ of our ARROneNS$_\Delta$ (DC) can also be roughly approximated by 60 Mbits (3 seconds) by replacing μ with $\mu e^{-\varepsilon_1}$ in (10).

4-Cycle Trick. We also investigated the effectiveness of our 4-cycle trick in ARROneNS$_\Delta$ and ARRTwoNS$_\Delta$ in detail. To this end, we evaluated our three algorithms when we did *not* add the Laplacian noise at the second round. Note that they do not provide edge LDP, as $\varepsilon_2 = \infty$. The purpose here is to purely investigate the effectiveness of the 4-cycle trick related to our first research question RQ1.

Figure 9 shows the results, where ε_1 and μ^* are changed to various values. Figure 9 shows that ARROneNS$_\Delta$ and ARRTwoNS$_\Delta$ significantly outperform ARRFull$_\Delta$ when μ^* is small.

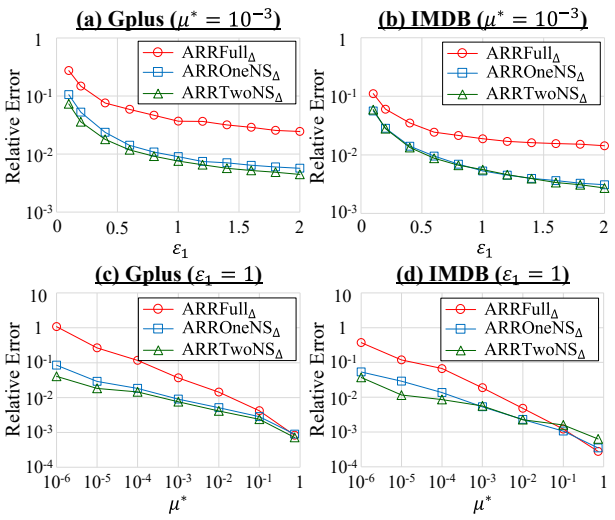


Figure 9: Relative error of our three algorithms without the Laplacian noise ($n = 107614$ in Gplus, $n = 896308$ in IMDB).

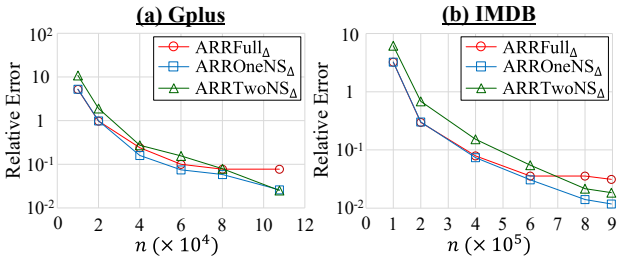


Figure 10: Relative error of our three algorithms with double clipping for various values of n ($\varepsilon = 1$, $\mu^* = 10^{-3}$).

This is because in both ARROneNS$_\triangle$ and ARRTwoNS$_\triangle$, the factors of C_4 (#4-cycles) and S_3 (#3-stars) in the expected l_2 loss diminish for small μ, as explained in Section 4.3. In other words, ARROneNS$_\triangle$ and ARRTwoNS$_\triangle$ effectively address the 4-cycle issue. Figure 9 also shows that ARRTwoNS$_\triangle$ slightly outperforms ARROneNS$_\triangle$ when μ^* is small. This is because the factors of C_4 and S_3 diminish more rapidly; i.e., ARRTwoNS$_\triangle$ addresses the 4-cycle issue more aggressively.

However, when we add the Laplacian noise, ARRTwoNS$_\triangle$ (DC) is outperformed by ARROneNS$_\triangle$ (DC), as shown in Figure 8. This is because ARRTwoNS$_\triangle$ cannot effectively reduce the global sensitivity by double clipping. In Figure 8, the difference between ARROneNS$_\triangle$ (DC) and ARRFull$_\triangle$ (DC) is also small for very small ε or μ^* (e.g., $\varepsilon = 0.1$, $\mu^* = 10^{-6}$) because the Laplacian noise is dominant in this case.

Changing n. We finally evaluated our three algorithms with double clipping while changing the number n of users. In both Gplus and IMDB, we randomly selected n users from all users and extracted a graph with n users. Then we evaluated the relative error while changing n to various values.

Figure 10 shows the results, where $\varepsilon = 1$ ($\varepsilon_0 = 0.1$, $\varepsilon_1 = \varepsilon_2 = 0.45$) and $\mu^* = 10^{-3}$. In all three algorithms, the relative error decreases with increase in n. This is because the expected l_2 loss can be expressed as $O(nd_{max}^3)$ or $O(nd_{max}^2)$ in these algorithms as shown in Section 5.3 and the square of the true triangle count can be expressed as $\Omega(n^2)$. In other words, when $d_{max} \ll n$, the relative error becomes smaller for larger n. Figure 10 also shows that for small n, ARRTwoNS$_\triangle$ provides the worst performance and ARROneNS$_\triangle$ performs almost the same as ARRFull$_\triangle$. For large n, ARRFull$_\triangle$ performs the worst and ARROneNS$_\triangle$ performs the best.

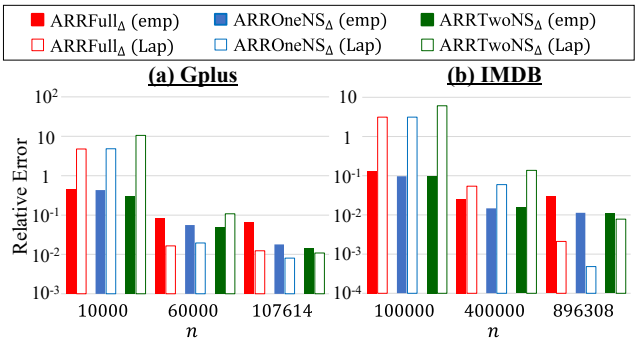


Figure 11: Relative error of empirical estimation and the Laplacian noise in our three algorithms with double clipping ($\varepsilon = 1$, $\mu^* = 10^{-3}$).

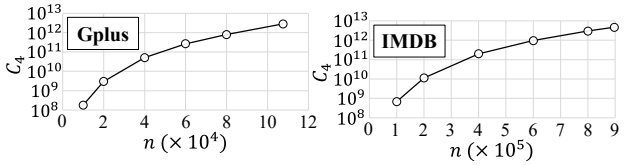


Figure 12: #4-cycles C_4.

To investigate the reason for this, we decomposed the estimation error into two components – the first error caused by empirical estimation and the second error caused by the Laplacian noise. Specifically, for each algorithm, we evaluated the first error by calculating the relative error when we did not add the Laplacian noise ($\varepsilon_1 = 0.45$). Then we evaluated the second error by subtracting the first error from the relative error when we used double clipping ($\varepsilon_0 = 0.1$, $\varepsilon_1 = \varepsilon_2 = 0.45$).

Figure 11 shows the results for some values of n, where "emp" represents the first error by empirical estimation and "Lap" represents the second error by the Laplacian noise. We observe that the second error rapidly decreases with increase in n. In addition, the first error of ARRFull$_\triangle$ is much larger than those of ARROneNS$_\triangle$ and ARRTwoNS$_\triangle$ when n is large.

We also examined the number C_4 of 4-cycles as a function of n. Figure 12 shows the results. We observe that C_4 (which is $O(nd_{max}^3)$) is quartic in n; e.g., C_4 is increased by $2^4 \approx 10$ and $6^4 \approx 10^3$ when n is multiplied by 2 and 6, respectively. This is because we randomly selected n users from all users and d_{max} is almost proportional to n (though $d_{max} \ll n$).

Based on Figures 11 and 12, we can explain Figure 10 as follows. As shown in Section 5.3, the l_2 loss of empirical estimation can be expressed as $O(nd_{max}^3)$, $O(nd_{max}^2)$, and

$O(nd_{max}^2)$ in ARRFull$_\triangle$, ARROneNS$_\triangle$, and ARRTwoNS$_\triangle$, respectively. The large l_2 loss of ARRFull$_\triangle$ is caused by a large value of C_4. The expected l_2 loss of the Laplacian noise is $O(\sum_{i=1}^{n} \kappa_i^2)$, which is much smaller than $O(nd_{max}^2)$. Thus, as n increases, the Laplacian noise becomes relatively very small, as shown in Figure 11. Consequently, ARROneNS$_\triangle$ provides the best performance for large n because it addresses the 4-cycle issue and effectively reduces the global sensitivity. This explains the results in Figure 10. It is also interesting that when $n \approx 10^5$, ARRFull$_\triangle$ performs the worst in Gplus and almost the same as ARROneNS$_\triangle$ in IMDB (see Figure 10). This is because Gplus is more dense than IMDB and C_4 is much larger in Gplus when $n \approx 10^5$, as in Figure 12.

In other words, Figures 10, 11, and 12 are consistent with our theoretical results in Section 5.3. From these results, we conclude that ARROneNS$_\triangle$ is effective especially for a large graph (e.g., $n \approx 10^6$) or dense graph (e.g., Gplus) where the number C_4 of 4-cycles is large.

Summary. In summary, our answers to our three research questions RQ1-3 are as follows. [RQ1]: Our ARROneNS$_\triangle$ achieves almost the smallest estimation error in all cases and outperforms the other two, especially for a large graph or dense graph where C_4 is large. [RQ2]: Our double clipping reduces the estimation error by two or three orders of magnitude. [RQ3]: Our entire algorithm (ARROneNS$_\triangle$ with double clipping) dramatically reduces the communication cost, e.g., from 6 hours to 8 seconds or less (relative error $= 0.21$) in IMDB at a 20 Mbps download rate [7].

Thus, triangle counting under edge LDP is now much more practical. In Appendix C, we show that the clustering coefficient can also be accurately estimated using our algorithms.

7 Conclusions

We proposed triangle counting algorithms under edge LDP with a small estimation error and small communication cost. We showed that our entire algorithms with the 4-cycle trick and double clipping dramatically reduce the download cost of [29], e.g., from 6 hours to 8 seconds or less.

We assumed that each user v_i honestly inputs her neighbor list $\mathbf{a}_i$, as in most previous work on LDP. However, recent studies [17, 20] show that the estimate in LDP can be skewed by data poisoning attacks. As future work, we would like to analyze the impact of data poisoning on our algorithms and develop defenses (e.g., detection) against it.

Acknowledgments

Kamalika Chaudhuri and Jacob Imola would like to thank ONR under N00014-20-1-2334 and UC Lab Fees under LFR 18-548554 for research support. Takao Murakami was supported in part by JSPS KAKENHI JP19H04113.

References

[1] Tools: TriangleLDP. https://github.com/TriangleLDP/TriangleLDP.

[2] 12th Annual Graph Drawing Contest. http://mozart.diei.unipg.it/gdcontest/contest2005/index.html, 2005.

[3] What to Do When Your Facebook Profile is Maxed Out on Friends. https://authoritypublishing.com/social-media/what-to-do-when-your-facebook-profile-is-maxed-out-on-friends/, 2012.

[4] The diaspora* project. https://diasporafoundation.org/, 2021.

[5] Mastodon: Giving social networking back to you. https://joinmastodon.org/, 2021.

[6] Minds: The leading alternative social network. https://wefunder.com/minds, 2021.

[7] YouTube: System requirements. https://support.google.com/youtube/answer/78358?hl=en, 2021.

[8] J. Acharya, Z. Sun, and H. Zhang. Hadamard response: Estimating distributions privately, efficiently, and with little communication. In *Proc. AISTATS'19*, pages 1120–1129, 2019.

[9] G. Andrew, O. Thakkar, H. B. McMahan, and S. Ramaswamy. Differentially private learning with adaptive clipping. In *Proc. NeurIPS'21*, pages 1–12, 2021.

[10] R. Arratia and L. Gordon. Tutorial on large deviations for the binomial distribution. *Bulletin of Mathematical Biology*, 51(1):125–131, 1989.

[11] A. L. Barabási. *Network Science*. Cambridge University Press, 2016.

[12] R. Bassily, K. Nissim, U. Stemmer, and A. Thakurta. Practical locally private heavy hitters. In *Proc. NIPS'17*, pages 2285—-2293, 2017.

[13] S. K. Bera and A. Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *Proc. STACS'17*, pages 11:1–11:14, 2017.

[14] S. K. Bera and C. Seshadhri. How the degeneracy helps for triangle counting in graph streams. In *Proc. PODS'20*, pages 457–467, 2020.

[15] S. K. Bera and C. Seshadhri. How to count triangles, without seeing the whole graph. In *Proc. KDD'20*, pages 306–316, 2020.

[16] V. Bindschaedler and R. Shokri. Synthesizing plausible privacy-preserving location traces. In *Proc. S&P'16*, pages 546–563, 2016.

[17] X. Cao, J. Jia, and N. Z. Gong. Data poisoning attacks to local differential privacy protocols. In *Proc. Usenix Security'21*, pages 947–964, 2021.

[18] R. Chan. The cambridge analytica whistleblower explains how the firm used facebook data to sway elections. https://www.businessinsider.com/cambridge-analytica-whistleblower-christopher-wylie-facebook-data-2019-10, 2019.

[19] R. Chen, G. Acs, and C. Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *Proc. CCS'12*, pages 638–649, 2012.

[20] A. Cheu, A. Smith, and J. Ullman. Manipulation attacks in local differential privacy. In *Proc. S&P'21*, pages 883–900, 2021.

[21] S. Chu and J. Cheng. Triangle listing in massive networks and its applications. In *Proc. KDD'11*, pages 672–680, 2020.

[22] E. Cyffers and A. Bellet. Privacy amplification by decentralization. *CoRR*, 2012.05326, 2021.

[23] W. Y. Day, N. Li, and M. Lyu. Publishing graph degree distribution with node differential privacy. In *Proc. SIGMOD'16*, pages 123–138, 2016.

[24] X. Ding, S. Sheng, H. Zhou, X. Zhang, Z. Bao, P. Zhou, and H. Jin. Differentially private triangle counting in large graphs. *IEEE Transactions on Knowledge and Data Engineering (Early Access)*, pages 1–14, 2021.

[25] C. Dwork and A. Roth. *The Algorithmic Foundations of Differential Privacy*. Now Publishers, 2014.

[26] T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. In *Proc. FOCS'15*, pages 614–633, 2015.

[27] U. Erlingsson, V. Pihur, and A. Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In *Proc. CCS'14*, pages 1054–1067, 2014.

[28] M. Hay, C. Li, G. Miklau, and D. Jensen. Accurate estimation of the degree distribution of private networks. In *Proc. ICDM'09*, pages 169–178, 2009.

[29] J. Imola, T. Murakami, and K. Chaudhuri. Locally differentially private analysis of graph statistics. In *Proc. USENIX Security'21*, pages 983–1000, 2021.

[30] J. Imola, T. Murakami, and K. Chaudhuri. Communication-efficient triangle counting under local differential privacy. *CoRR*, 2110.06485, 2022.

[31] Z. Jorgensen, T. Yu, and G. Cormode. Publishing attributed social graphs with formal privacy guarantees. In *Proc.SIGMOD'16*, pages 107–122, 2016.

[32] M. Joseph, J. Mao, and A. Roth. Exponential separations in local differential privacy. In *Proc. SODA'20*, pages 515–527, 2020.

[33] P. Kairouz, K. Bonawitz, and D. Ramage. Discrete distribution estimation under local privacy. In *Proc. ICML'16*, pages 2436–2444, 2016.

[34] P. Kairouz, H. B. McMahan, and B. Avent *et al.* Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2):1–210, 2021.

[35] J. Kallaugher, A. McGregor, E. Price, and S. Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In *Proc. PODS'19*, pages 119–133, 2019.

[36] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. *Proceedings of the VLDB Endowment*, 4(11):1146–1157, 2011.

[37] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In *Proc. TCC'13*, pages 457–476, 2013.

[38] A. Kolluri, T. Baluta, and P. Saxena. Private hierarchical clustering in federated networks. In *Proc. CCS'21*, pages 2342–2360, 2021.

[39] N. Li, M. Lyu, and D. Su. *Differential Privacy: From Theory to Practice*. Morgan & Claypool Publishers, 2016.

[40] M. Manjunath, K. Mehlhorn, K. Panagiotou, and H. Sun. Approximate counting of cycles in streams. In *Proc. ESA'11*, pages 677–688, 2011.

[41] J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *Proc. NIPS'12*, pages 539–547, 2012.

[42] A. McGregor and S. Vorotnikova. Triangle and four cycle counting in the data stream model. In *Proc. PODS'20*, pages 445–456, 2020.

[43] C. Morris. The number of data breaches in 2021 has already surpassed last year's total. https://fortune.com/2021/10/06/data-breach-2021-2020-total-hacks/, 2021.

[44] T. Murakami and Y. Kawamoto. Utility-optimized local differential privacy mechanisms for distribution estimation. In *Proc. USENIX Security'19*, pages 1877–1894, 2019.

[45] M. E. J. Newman. Random graphs with clustering. *Physical Review Letters*, 103(5):058701, 2009.

[46] H. H. Nguyen, A. Imine, and M. Rusinowitch. Network structure release under differential privacy. *Transactions on Data Privacy*, 9(3):215–214, 2016.

[47] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proc. STOC'07*, pages 75–84, 2007.

[48] T. Paul, A. Famulari, and T. Strufe. A survey on decentralized online social networks. *Computer Networks*, 75:437–452, 2014.

[49] Z. Qin, T. Yu, Y. Yang, I. Khalil, X. Xiao, and K. Ren. Generating synthetic decentralized social graphs with local differential privacy. In *Proc. CCS'17*, pages 425–438, 2017.

[50] S. Raskhodnikova and A. Smith. Efficient lipschitz extensions for high-dimensional graph statistics and node private degree distributions. *CoRR*, 1504.07912, 2015.

[51] S. Raskhodnikova and A. Smith. *Differentially Private Analysis of Graphs*, pages 543–547. Springer, 2016.

[52] G. Robins, P. Pattison, Y. Kalish, and D. Lusher. An introduction to exponential random graph (p^*) models for social networks. *Social Networks*, 29(2):173–191, 2007.

[53] C. Sabater, A. Bellet, and J. Ramon. An accurate, scalable and verifiable protocol for federated differentially private averaging. *CoRR*, 2006.07218, 2021.

[54] C. Seshadhri, A. Pinar, and T. G. Kolda. Triadic measures on graphs: The power of wedge sampling. In *Proc. SDM'13*, pages 10–18, 2013.

[55] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *Proc. CCS'15*, pages 1310–1321, 2015.

[56] H. Sun, X. Xiao, I. Khalil, Y. Yang, Z. Qui, H. Wang, and T. Yu. Analyzing subgraph statistics from extended local views with decentralized differential privacy. In *Proc. CCS'19*, pages 703–717, 2019.

[57] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proc. WWW'11*, pages 607–614, 2011.

[58] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. DOULION: Counting triangles in massive graphs with a coin. In *Proc. KDD'09*, pages 837–846, 2009.

[59] C. E. Tsourakakis, M. N. Kolountzakis, and G. L. Miller. Triangle sparsifiers. *Journal of Graph Algorithms and Applications*, 15(6):703–726, 2011.

[60] T. Wang, J. Blocki, N. Li, and S. Jha. Locally differentially private protocols for frequency estimation. In *Proc. USENIX Security'17*, pages 729–745, 2017.

[61] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[62] B. Wu, K. Yi, and Z. Li. Counting triangles in large graphs by random sampling. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2013–2026, 2016.

[63] X. Xiao, G. Bender, M. Hay, and J. Gehrke. iReduct: Differential privacy with reduced relative errors. In *Proc. SIGMOD'11*, pages 229–240, 2011.

[64] Q. Ye, H. Hu, M. H. Au, X. Meng, and X. Xiao. Towards locally differentially private generic graph metric estimation. In *Proc. ICDE'20*, pages 1922–1925, 2020.

[65] Q. Ye, H. Hu, M. H. Au, X. Meng, and X. Xiao. LF-GDPR: A framework for estimating graph metrics with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering (Early Access)*, pages 1–16, 2021.

[66] H. Zhang, S. Latif, R. Bassily, and A. Rountev. Differentially-private control-flow node coverage for software usage analysis. In *Proc. USENIX Security'20*, pages 1021–1038, 2020.

[67] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Private release of graph statistics using ladder functions. In *Proc. SIGMOD'15*, pages 731–745, 2015.

A Basic Notations

Table 2 shows the basic notations used in this paper.

B Comparison with One-Round Algorithms

Below we show that one-round triangle counting algorithms suffer from a prohibitively large estimation error.

First, we note that all of the existing one-round triangle algorithms in [29,64,65] are inefficient and *cannot be directly applied to a large-scale graph* such as Gplus and IMDB in Section 6. Specifically, in their algorithms, each user v_i applies Warner's RR to each bit of her neighbor list $\mathbf{a}_i$ and sends the noisy neighbor list to the server. Then the server counts the number of noisy triangles, each of which has three noisy edges, and estimates $f_\triangle(G)$ based on the noisy triangle count. The noisy graph G' in the server is dense, and there are $O(n^3)$ noisy triangles in G'. Thus, the time complexity of the existing

Table 2: Basic notations.

Symbol	Description
$G = (V, E)$	Graph with n users V and edges E.
v_i	i-th user in V (i.e., $V = \{v_1, \ldots, v_n\}$).
d_{max}	Maximum degree of G.
$\mathcal{G}$	Set of possible graphs with n users.
$f_\triangle(G)$	Triangle count in G.
$\mathbf{A} = (a_{i,j})$	Adjacency matrix.
$\mathbf{a}_i$	Neighbor list of v_i (i.e., i-th row of $\mathbf{A}$).
$\mathcal{R}_i$	Local randomizer of v_i.
M_i	Message sent from the server to user v_i.
μ	Parameter in the ARR.
μ^*	$= \mu, \mu^2, \mu^3$ in ARRFull$_\triangle$, ARROneNS$_\triangle$, and ARRTwoNS$_\triangle$, respectively.
$\tilde{d}_i$	Noisy degree of user v_i.
κ_i	Clipping threshold of user v_i.
ε_0	Privacy budget for edge clipping.
ε_1	Privacy budget for the ARR.
ε_2	Privacy budget for the Laplacian noise.
ε	Total privacy budget.

one-round algorithms [29, 64, 65] is $O(n^3)$. It is also reported in [29] that when $n = 10^6$, the one-round algorithms would require about 35 years even using a supercomputer, due to the enormous number of noisy triangles.

Therefore, we evaluated the existing one-round algorithms by taking the following two steps. First, we evaluate all the existing algorithms in [29, 64, 65] using small graph datasets ($n = 10000$) and show that the algorithm in [29] achieves the lowest estimation error. Second, we improve the time complexity of the algorithm in [29] using the ARR (i.e., edge sampling after Warner's RR) and compare it with our two-rounds algorithms using large graph datasets in Section 6.

Small Datasets. We first evaluated the existing algorithms in [29, 64, 65] using small datasets. For both Gplus and IMDB in Section 6, we first randomly selected $n = 10000$ users from all users and extracted a graph with n users. Then we evaluated the relative error of the following three algorithms: (i) RR (biased) [29, 64], (ii) RR (bias-reduced) [65], and (iii) RR (unbiased) [29]. All of them provide ε-edge LDP.

RR (biased) simply uses the number of noisy triangles in the noisy graph G' obtained by Warner's RR as an estimate of $f_\triangle(G)$. Clearly, it suffers from a very large bias, as G' is dense. RR (bias-reduced) reduces this bias by using a noisy degree sent by each user. However, it introduces some approximation to estimate $f_\triangle(G)$, and consequently, it is not clear whether the estimate is unbiased. We used the mean of the noisy degrees as a representative degree to obtain the optimal privacy budget allocation (see [65] for details). RR (unbiased) calculates an unbiased estimate of $f_\triangle(G)$ via empirical estimation. It is proved that the estimate is unbiased [29].

In all of the three algorithms, each user v_i obfuscates bits

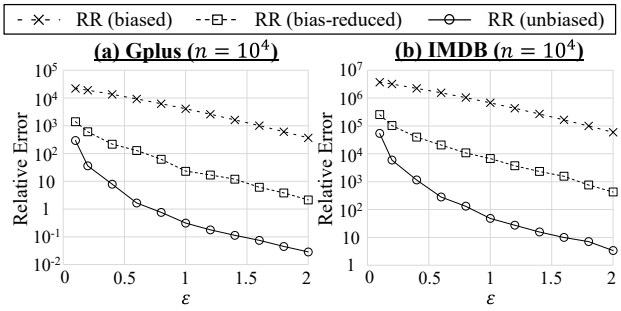


Figure 13: Relative error of one-round algorithms for small datasets ($n = 10000$).

for smaller user IDs in her neighbor list $\mathbf{a}_i$. We averaged the relative error over 10 runs.

Figure 13 shows the results. RR (bias-reduced) significantly outperforms RR (biased) and is significantly outperformed by RR (unbiased). We believe this is caused by the fact that RR (bias-reduced) introduces some approximation and does not calculate an unbiased estimate of $f_\triangle(G)$.

Large Datasets. Based on Figure 13, we improve the time complexity of RR (unbiased) using the ARR and compare it with our two-rounds algorithms in large datasets.

Specifically, RR (unbiased) counts *triangles*, 2-*edges* (three nodes with two edges), 1-*edges* (three nodes with one edge), and *no-edges* (three nodes with no edges) in G' obtained by Warner's RR. Let $m_3, m_2, m_1, m_0 \in \mathbb{Z}_{\geq 0}$ be the numbers of triangles, 2-edges, 1-edges, and no-edges, respectively, after applying Warner's RR. RR (unbiased) calculates an unbiased estimate of $f_\triangle(G)$ from these four values. Thus, we improve RR (unbiased) by using the ARR, which samples each edge with probability p_2 after Warner's RR, and then calculating unbiased estimates of m_3, m_2, m_1, and m_0.

Let $\hat{m}_3, \hat{m}_2, \hat{m}_1, \hat{m}_0 \in \mathbb{R}$ be the unbiased estimates of m_3, m_2, m_1, and m_0, respectively. Let $m_3^*, m_2^*, m_1^*, m_0^* \in \mathbb{Z}_{\geq 0}$ be the number of triangles, 2-edges, 1-edges, no-edges, respectively, after applying the ARR. Since the ARR samples each edge with probability p_2, we obtain:

$$
\begin{aligned}
m_3^* &= p_2^3 \hat{m}_3 \\
m_2^* &= 3p_2^2(1-p_2)\hat{m}_3 + p_2^2 \hat{m}_2 \\
m_1^* &= 3p_2(1-p_2)^2 \hat{m}_3 + 2p_2(1-p_2)\hat{m}_2 + p_2 \hat{m}_1.
\end{aligned}
$$

By these equations, we obtain:

$$\hat{m}_3 = \frac{m_3^*}{p_2^3} \tag{15}$$

$$\hat{m}_2 = \frac{m_2^*}{p_2^2} - 3(1-p_2)\hat{m}_3 \tag{16}$$

$$\hat{m}_1 = \frac{m_1^*}{p_2} - 3(1-p_2)^2 \hat{m}_3 - 2(1-p_2)\hat{m}_2 \tag{17}$$

$$\hat{m}_0 = \frac{n(n-1)(n-2)}{6} - \hat{m}_3 - \hat{m}_2 - \hat{m}_1. \tag{18}$$

Therefore, after applying the ARR to the lower triangular part of $\mathbf{A}$, the server counts m_3^*, m_2^*, m_1^*, and m_0^* in G', and

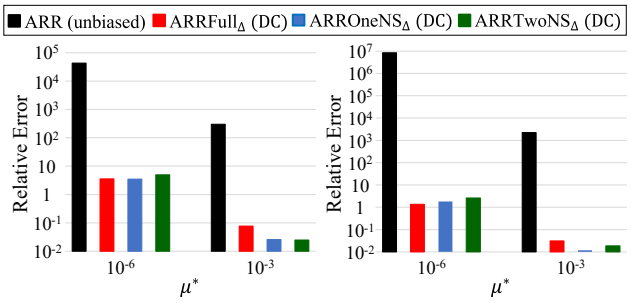


Figure 14: Relative error of the one-round algorithm ARR (unbiased) and our three two-rounds algorithms with double clipping for large datasets ($n = 107614$ in Gplus, $n = 896308$ in IMDB).

then calculates the unbiased estimates $\hat{m}_3$, $\hat{m}_2$, $\hat{m}_1$, and $\hat{m}_0$ by (15), (16), (17), and (18), respectively. Finally, the server estimates $f_\triangle(G)$ from $\hat{m}_3$, $\hat{m}_2$, $\hat{m}_1$, and $\hat{m}_0$ in the same way as RR (unbiased). We denote this algorithm by ARR (unbiased). The time complexity of ARR (unbiased) is $O(\mu^3 n^3)$, where μ is the ARR parameter.

We compared ARR (unbiased) with our three algorithms with double clipping using Gplus ($n = 107614$) and IMDB ($n = 896308$). For the sampling probability p_2, we set $p_2 = 10^{-3}$ or 10^{-6}. We averaged the relative error over 10 runs.

Figure 14 shows the results, where we set $\mu^* = 10^{-6}$ or 10^{-3}. In ARR (unbiased), we used μ^* as the ARR parameter μ. Thus, we can see *how much the relative error is reduced by introducing an additional round with ARRFull$_\triangle$*. Figure 14 shows that the relative error of ARR (unbiased) is prohibitively large; i.e., relative error $\gg 1$. This is because three edges are noisy in any noisy triangle. The relative error is significantly reduced by introducing an additional round because only one edge is noisy in each noisy triangle at the second round.

In summary, one-round algorithms are far from acceptable in terms of the estimation error for large graphs, and two-round algorithms such as ours are necessary.

C Clustering Coefficient

Here we evaluate the estimation error of the clustering coefficient using our algorithms.

We first estimated a triangle count by using our ARROneNS$_\triangle$ with double clipping ($\varepsilon_0 = \frac{\varepsilon}{10}$ and $\varepsilon_1 = \varepsilon_2 = \frac{9\varepsilon}{20}$) because it provides the best performance in Figures 7, 8, and 10. Then we estimated a 2-star count by using the one-round 2-star algorithm in [29] with the edge clipping in Section 5.

Specifically, we calculated a noisy degree $\tilde{d}_i$ of each user v_i by using the edge clipping with the privacy budget ε_0. Then we calculated the number $r_i \in \mathbb{Z}_{\geq 0}$ of 2-stars of which user v_i is a center, and added $\mathrm{Lap}(\frac{\Delta}{\varepsilon_1})$ to r_i, where $\Delta = \binom{\tilde{d}_i}{2}$. Let $\hat{r}_i = r_i + \mathrm{Lap}(\frac{\Delta}{\varepsilon_1})$ be the noisy 2-star of v_i. Finally, we calculated the sum $\sum_{i=1}^{n} \hat{r}_i$ as an estimate of the 2-star count.

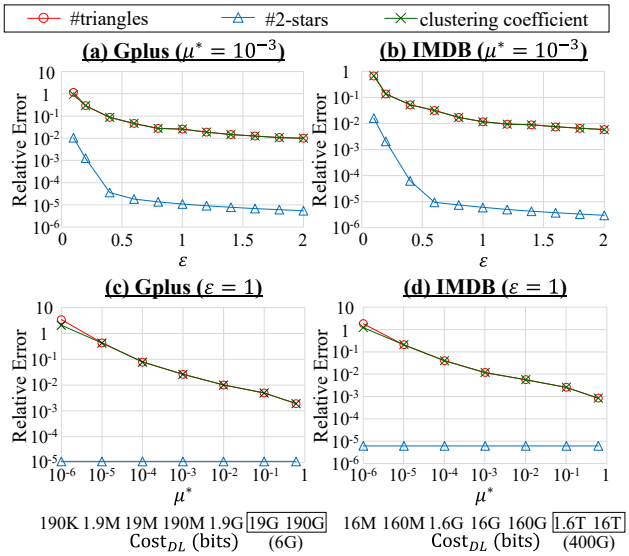


Figure 15: Relative errors of #triangles, #2-stars, and the clustering coefficient in ARROneNS$_\triangle$ with double clipping. Cost$_{DL}$ is calculated by (10) (when $\mu^* \geq 0.1$, Cost$_{DL}$ can be 6 Gbits and 400 Gbits in Gplus and IMDB, respectively).

This 2-star algorithm provides $(\varepsilon_0 + \varepsilon_1)$-edge privacy (see [29] for details). For the privacy budgets ε_0 and ε_1, we set $\varepsilon_0 = \frac{\varepsilon}{10}$ and $\varepsilon_1 = \frac{9\varepsilon}{10}$.

Based on the triangle and 2-star counts, we estimated the clustering coefficient as $\frac{3 \times \hat{f}_\triangle(G)}{\hat{f}_{2\star}(G)}$, where $\hat{f}_\triangle(G)$ (resp. $\hat{f}_{2\star}(G)$) is the estimate of the triangle (resp. 2-star) count.

Figure 15 shows the relative errors of the triangle count, 2-star count, and clustering coefficient. Note that the relative error of the 2-star count is not changed by changing μ^* because the 2-star algorithm does not use the ARR. Figure 15 shows that the relative error of the 2-star count is much smaller than that of the triangle count. This is because each user can count her 2-stars locally (whereas she cannot count her triangles), as described in Section 1. Consequently, the relative error of the clustering coefficient is almost the same as that of the triangle count, as the denominator $\hat{f}_{2\star}(G)$ in the clustering coefficient is very accurate.

Note that the clustering coefficient requires the privacy budgets for calculating both $\hat{f}_\triangle(G)$ and $\hat{f}_{2\star}(G)$ (in Figure 15, 2ε in total). However, we can accurately calculate $\hat{f}_{2\star}(G)$ with a very small privacy budget, as shown in Figure 15. Thus, we can accurately estimate the clustering coefficient with almost the same privacy budget as the triangle count by assigning a very small privacy budget (e.g., $\varepsilon = 0.1$ or 0.2) for $\hat{f}_{2\star}(G)$.

In summary, we can accurately estimate the clustering coefficient as well as the triangle count under edge LDP by using our ARROneNS$_\triangle$ with double clipping.