Differentially-Private Network Trace Analysis

Frank McSherry Ratul Mahajan

Microsoft Research

Abstract– We consider the potential for network trace analysis while providing the guarantees of "differential privacy." While differential privacy provably obscures the presence or absence of individual records in a dataset, it has two major limitations: analyses must (presently) be expressed in a higher level declarative language; and the analysis results are randomized before returning to the analyst.

We report on our experiences conducting a diverse set of analyses in a differentially private manner. We are able to express all of our target analyses, though for some of them an approximate expression is required to keep the error-level low. By running these analyses on real datasets, we find that the error introduced for the sake of privacy is often (but not always) low even at high levels of privacy. We factor our learning into a toolkit that will be likely useful for other analyses. Overall, we conclude that differential privacy shows promise for a broad class of network analyses.

Categories and Subject Descriptors
C.2.m [Computer-communication networks] Miscellaneous

General Terms
Algorithms, experimentation, measurement

Keywords
Differential privacy, trace analysis

1. INTRODUCTION

As a community, if we do not solve this problem [privacy-compliant data sharing], we are in trouble.
– Vern Paxson (HotNets-VIII, 2009)

The complexity of modern networks makes access to real-world data critical to networking research. Without this access it is almost impossible to understand how the network behaves and how well a proposed enhancement will function if deployed. But obtaining relevant data today is a highly frustrating exercise for researchers and one that can often end in failure.

Thus far, the community has mainly taken the social approach of encouraging institutions and researchers to release collected data (e.g., CRAWDAD [6], ITA [11]). While beneficial, this approach has weaknesses. The released data is heavily sanitized (e.g., payloads are removed) and anonymized, limiting their research value [21]. Worse, as demonstrated by research [5, 26, 21] and real mishaps [2, 29, 20], anonymization is vulnerable to attacks that infer sensitive information. Because of this fear, many data owners today prefer the safer option of not releasing data at all.

Consider an alternative approach to enable data-driven networking research: instead of releasing sanitized data, the data owners run analyses on behalf of the researchers; to preserve privacy, restrictions are placed on what analyses are permitted and what output is returned. This approach was first advocated by Mogul and Arlitt [19] and recently termed *mediated trace analysis* by Mittal et al. [18].

Given the intricacies of protecting sensitive information and past failures, we believe that strong and formal privacy guarantees are an important prerequisite for data owners to adopt this approach. Existing proposals, however, provide no guarantee. The basis for protecting privacy in Mogul and Arlitt's original proposal is human verification, which is error-prone and hard to scale to sophisticated analyses [19]. To obviate human verification, Mirkovic proposes rules that an analysis must follow to protect privacy [17]. It is unclear, however, what privacy properties are achieved by these rules. Mittal et al. propose that only analyses that leak fewer than a threshold number of bits (in an information-theoretic sense) be allowed [18]. However, restricting information leakage and preserving privacy are not the same. An analysis that reveals if hosts A and B communicate leaks only one bit but may represent an unacceptable privacy loss for the hosts.

We ask if mediated trace analysis can be enabled with formal privacy guarantees. The definition of privacy that we consider is differential privacy [8, 7]. Informally, differential privacy guarantees that the presence or absence of individual records is hard to infer from the analysis output. While it is unclear if differential privacy is the appropriate guarantee for networking analyses—or if there even exists a single definition that applies to all analyses and datasets—we consider it because it provides one of the strongest known privacy guarantees. Appealingly, it is resilient to collusion, supports multiple interactive queries, and is also independent of any auxiliary information that an attacker might possess; such information has been shown to break anonymization [5, 26, 20, 29]. As such, differential privacy has the potential to provide a strong foundation for mediated data analysis.

However, the strong guarantees of differential privacy do not come for free. Privacy is preserved by adding noise to the output of the analysis, imposing on its accuracy. The added noise is scaled to mask the presence or absence of small sets of records. While the magnitude of the noise is typically small, and the distribution of the noise is known to the analyst, it can render sensitive analyses useless. Additionally, using current tools a differentially-private analysis

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.
Copyright 2010 ACM 978-1-4503-0201-2/10/08 ...$10.00.

must be expressed using high-level operations (e.g., SQL-like) on the data, so that the privacy-preserving platform can understand how the analysis manipulates data and add noise accordingly.

Given limitations of accuracy and expressibility, the questions of whether and which networking analyses can be fruitfully conducted in a differentially private manner is open. The answers depend both on the nature of the analyses and the data. Differential privacy is a recent development, and its practical utility is still unclear, even outside of networking. We are aware of only two concrete case studies [15, 24], and the results are mixed.

To shed light on the possibility of network trace analysis with differential privacy guarantees, we attempt to reproduce a spectrum of network trace analyses using PINQ [14], a differentially-private analysis platform. Our analyses include multiple examples of packet-level, flow-level, and graph-level computations chosen from the networking literature. Each analysis relies on sensitive fields in the source data and will thus be difficult to conduct for researchers that do not own the data.

We find that we are able to express all the analyses that we consider, though some required approximations. Certain computations, such as arbitrary resolution cumulative distribution function, are fundamentally impossible with differential privacy (independent of the platform), but can be approximated with noisy counterparts. Certain others, such as sliding window computations and splitting a long flow into individual connections, are hard to implement in a manner that incurs only a small amount of noise. We find that the impact of our approximations on the results is low, however.

There are multiple ways to implement an analysis, with different privacy costs (i.e., added noise). We find that sometimes there is also a trade-off between algorithmic complexity and privacy cost. These challenges are surmountable, but they complicate (or, enrich) the task of implementing networking analyses. We implement a toolkit with analysis primitives that we find common to multiple analyses. To aid other researchers, we are releasing this toolkit and our analysis implementations [23].

We find that the added noise tends to not be a hindrance because most analyses seek only broad distributional and statistical information about the data. They rarely depend heavily on few individual records, and differential privacy is, in principle, compatible with this use. The main challenge lies in extracting sufficient aggregates from the data in a privacy-efficient manner. For a few analyses, we achieve high accuracy only when the privacy level is low. As we gain more experience at implementing privacy-preserving analyses, this situation should only improve.

Overall, we conclude that differential privacy is a promising avenue for enabling mediated trace analysis for a large class of analyses. Our work, however, is only the first step. Before we can start convincing data owners to share data, we need to resolve several key issues. One is managing privacy loss due to repeat analysis of the same data. Another is preserving privacy, with acceptable analysis noise, for higher-level entities (e.g., hosts, subnets) that may be spread across many records. Yet another issue is developing guidelines regarding appropriate privacy levels for various situations. Building on the strong foundation that is provided by differential privacy, we hope that future work can resolve these issues to the satisfaction of many data owners.

2. BACKGROUND

In this section, we give a brief background on differential privacy and contrast it with alternative privacy definitions. We also describe PINQ, the analysis platform we use in our investigation.

2.1 Differential Privacy

Differential privacy requires that a computation exhibit essentially identical behavior on two data sets that differ only in a small number of records. Formally, let A and B be two datasets and $A \ominus B$ be the set of records in exactly one of them. Then, a randomized computation M provides ϵ-differential privacy if for all A and B and any subset S of the outputs of the computation:

$$\mathbf{Pr}[M(A) \in S] \quad \leq \quad \mathbf{Pr}[M(B) \in S] \times \exp(\epsilon |A \ominus B|)$$

That is, the probability of any consequence of the computation is almost independent of whether any one record is present in the input. For each record, it is almost as if the record was not used in the computation, a very strong baseline for privacy. The guarantee assumes that each record is independent of the rest and applies to all aspects of the record. So, if each record is a packet, differential privacy protects its IP addresses, payloads, ports, etc., as well as its very existence.

Differential privacy is preserved by adding "noise" to the outputs of a computation. Intuitively, this noise introduces uncertainty about the true value of the output, which translates into uncertainty about the true values of the inputs. The noise distributions that provide differential privacy vary as a function of the query, though most commonly we see Laplace noise (a symmetric exponential distribution). The magnitude of the noise is calibrated to the amount by which the output could change should a single input record arrive or depart, divided by ϵ. The value of a perturbed result depends greatly on the data, however; a count accurate to within ± 10 may be useful over a thousand records but not over ten records. The noise distribution is known to the analyst, who can judge if the noisy results are statistically significant or not without access to the actual data.

The parameter ϵ is a quantitative measurement of the strength of the privacy guarantee. Lower values correspond to stronger guarantees, with $\epsilon = 0$ being perfect privacy. Typically, $\epsilon \leq 0.1$ is considered strong and $\epsilon \geq 10$ is considered weak. We are not advocating specific levels of differential privacy as sufficient but are instead interested in understanding the trade-off between accuracy and privacy.

Comparison with alternative privacy definitions Unlike differential privacy, many alternative formulations do not provide a direct guarantee or are vulnerable to auxiliary information that the attacker might possess. Consider, for example, k-anonymity, which provides guidance on releasing data such that the identity of individual records remains private [29]. A release provides k-anonymity if the information for each record cannot be distinguished from at least k-1 other records. However, this definition provides no guarantee in the face of auxiliary information that may exist outside of the released dataset. Such information can break anonymization [20, 5, 26].

As another example, consider reducing information leakage as a way to preserve privacy [18]. The reasoning is that the fewer bits of information that an analysis leaks about specific records, the more privacy is protected. However,

Aggregations

Count	Std. deviation of added noise is $\sqrt{2}/\epsilon$.
Sum	Std. deviation of added noise is $\sqrt{2}/\epsilon$.
Average	Std. deviation of added noise is $\sqrt{8}/\epsilon n$, where n is the number of records.
Median	The return value partitions input into sets whose sizes differ by approx. $\sqrt{2}/\epsilon$

Transformations

Where, Select Distinct	No sensitivity increase
GroupBy	Increases sensitivity by two
Join, Concat Intersect	No sensitivity increase for either input
Partition	Privacy cost equals the maximum of the resulting partitions

Table 1: Main data operations in PINQ.

this reasoning is indirect at best and fallacious at worst. Revealing even one bit can lead to significant loss in privacy. For example, revealing if hosts A and B communicate requires only one bit of information but may represent an unacceptable loss in privacy. Moreover, any such scheme always leaks at least one bit, in response to: "did the analysis reveal too many bits?" This response bit can encode very sensitive information, and is always revealed to the analyst.

2.2 Privacy Integrated Queries (PINQ)

PINQ is an analysis platform that provides differential privacy [14]. Rather than provide direct access to the underlying data, PINQ provides an opaque `PINQueryable` object supporting various SQL-like operations. The analyst specifies queries over the data in a declarative language, and is rewarded with aggregate quantities that have been subjected to noise. Once a noised aggregate has been extracted from PINQ, it can be manipulated freely by the analyst, and used in further queries. PINQ tracks the privacy implications of successive operations and ensures that the cumulative privacy cost does not exceed a configured budget.

Table 1 summarizes the main data operations supported by PINQ and their privacy implications. There are two types of operations: aggregations and transformations. Aggregations return the aggregate value after adding noise per differential privacy. Transformations return a new `PINQueryable` object that can be further operated upon. They can amplify the sensitivity of subsequent queries, so that aggregations run with one value of ϵ may deplete many multiples of ϵ from the privacy budget. PINQ ensures that any amplification is properly accounted. Importantly, the logic within a transformation can act arbitrarily on the sensitive records.

The semantics of the transformations are similar to SQL, with two major exceptions. First, the `Join` operation in PINQ is not a standard equijoin, in which one record can match an unbounded number of other records. Instead, records in both data set are grouped by the key they are being joined on, so that the `Join` results in a list of pairs of groups. This restricts each pair to have limited impact on aggregates (that of a single record) despite being arbitrarily large, but it does enable differential privacy guarantees which would not otherwise exist.

A second difference is a `Partition` operation that can split a single protected data set into multiple protected data sets, using an arbitrary key selection function. This operation is important because the privacy cost to the source data set is the maximum of the costs to the multiple parts, rather than their sum. We can, for example, partition packets based on destination port, and conduct independent analyses on each part while costing only the maximum.

As the discussion above illustrates, and will become clearer later, the privacy cost of an analysis depends not only what the analysis aims to output but also on how it is expressed. PINQ is essentially a programming language, and the space of analyses that can be expressed is limited mainly by the analysts creativity. One of our contributions is to devise privacy-efficient ways of expressing network data analyses. We will see many common tools and programming patterns that we expect to be broadly useful, several of which we explicitly factor out into a re-usable toolkit.

2.3 An Example

Suppose we want to count distinct hosts that send more than 1024 bytes to port 80. This computation, which involves grouping packets by source and restricting the result based on what we see in each group, can be expressed as:[1]

```
packets = new PINQueryable<Packet>(trace, epsilon);
packets.Where(pkt => pkt.dstPort = 80)
       .GroupBy(pkt => pkt.srcIP)
       .Where(grp => grp.Sum(pkt => pkt.len) > 1024)
       .Count(epsilon_query);
```

The `Packet` type contains fields that we might expect, including sensitive fields such as IP addresses and payloads. The raw data lies in `trace`. The total privacy budget for the trace is `epsilon`, and the amount to be spent on this query is `epsilon_query`. The analyst can run multiple queries on the data as long as the total privacy cost is less than `epsilon`. The expressions of the form `x => f(x)` are anonymous functions that apply f to x.

For one of our datasets (the Hotspot trace in §3), the correct, noise-free answer for this analysis is 120. In a particular run with `epsilon`=0.1, we get an answer of 121. Different runs will yield different answers. The expected error for this analysis is ± 10.

3. DIFFERENTIALLY-PRIVATE NETWORK TRACE ANALYSIS

Our goal is to investigate if differential privacy can provide an effective basis for mediated trace analysis. If feasible, we can enable rich yet safe data analysis, without requiring the data owners to expose raw, anonymized, or sanitized data. As a precursor to conducting analysis, however, the analysts need to know the format of the stored data. This can be accomplished by having the data owners release format specifications or release synthetic data on which an analysis can be tested before submitting to the owner. A non-goal of our work is investigating if new analyses can be developed in a differentially private manner. This task, which is distinct from conducting existing analyses (or their variants), may require intimate access to raw data.

The strong and direct guarantees of differential privacy are appealing but its utility for network data analysis is

[1]The code fragments in this paper are stylized C# code. They will not compile or record outputs but are otherwise almost identical to actual PINQ code.

uncertain because of two issues. First, differential privacy introduces noise, which may incapacitate certain sensitive computations. Examples include arbitrary resolution CDFs and fragile statistics like minimum and maximum. Second, the analysis must (currently) be expressed in a restricted high-level language. Networking analyses are not typically constrained to such languages, and privacy aside it may be challenging to express analyses in such languages. These two constraints have interplay, in that the amount of noise introduced depends on how the analysis is expressed. We will see several cases where we must exchange fidelity to the original algorithm for a smaller amount of noise introduced.

The expressibility restriction could potentially be overcome by the invention of new differentially-private primitive computations. Although PINQ does contain mechanisms for extending the platform, the extensions become part of the trusted computing base. For this reason, we restrict our study to the existing operations supported by PINQ, to see how far we can go with just those operations. While we are largely successful, our experience does point at a few extensions that will be broadly useful.

To understand if differentially private network trace analysis is feasible, we consider a wide array of real analyses. We investigate the extent to which each can be faithfully expressed and its accuracy loss over real data.

Analyses Table 2 shows the analyses that we consider and summarizes our results (explained later). The analysis selection process was informal and intended to maximize diversity with a manageable number. We made a list of analyses that appear in recent networking literature and preferred those with computations that are disparate from others already picked. While picking an analysis, we ignore any prior expectations about whether it would be easy to conduct in a differentially private manner.

There is no standard classification of networking analyses to let us judge if we have included an analysis from each class. But based on our original list, we find that set of analyses can be classified as operating on the granularity of packets, flows, or graphs. As the table shows, our chosen set includes multiple examples of each category. That we can conduct these analyses in a differentially private manner does not imply that we can conduct *any* analysis. But the diversity of our selected analyses gives us confidence that if we can conduct these we can conduct a wide range of network trace analyses.

In addition to being diverse, these analyses require access to information that data owners typically consider sensitive. For instance, worm fingerprinting [27] (a packet-level analysis) requires raw packet payloads; stepping stone detection [33] (a flow-level analysis) requires addresses and ports in traffic flows; and anomaly detection [13] (a graph-level analysis) requires information on the amount of traffic at individual links of an ISP and how it varies across time. Because of the sensitivity of such information, researchers find it difficult today to conduct these and similar analyses on real data. If finding one data source for such analyses is difficult, finding multiple is almost impossible.

Datasets The analysis accuracy depends on the nature of the data. We thus use real network traces in our work. Table 3 shows the datasets that we study in this paper and the type and the number of records they contain. Different datasets are used by different analyses. The size of each is comparable to what its analysis typically operates on. We also studied other datasets [4, 11] for several of the analyses and obtained results similar to those presented below.

	Record	#records
Hotspot	<timestamp, packet>	7.0M
IspTraffic	<timestamp,link,packet>	15.7B
IPscatter	<monitor, IPaddr, ttl>	3.8M

Table 3: The datasets that we consider.

Hotspot is a *tcpdump* trace of packets that we collected on the wired access link of a large hotspot. It contains complete packets, including unaltered addresses and payloads.

IspTraffic is constructed from traffic at a large ISP (whose identity we are required to keep confidential). The ISP has over 400 links and it provided us highly aggregated information on traffic volume at each link in each 15-min window over a week-long period. We mimic a fine-grained dataset using this information by de-aggregating traffic volume into 1500-bytes packets that are spread evenly across the time window. Note that the aggregate representation of the source data is not itself a basis for differential privacy; the presence or absence of individual packets can still be observed in the precise aggregates.

IPscatter is a list of IP addresses and their TTL-distances from 38 monitors. It was constructed using the data collected by Spring et al. [28], who conducted *traceroute* probes from 38 PlanetLab sites to an IP address inside each BGP prefix. The constructed dataset includes a record for each IP seen along each probe.

Privacy level The accuracy of a differentially private analysis depends on the desired strength of the privacy guarantee (parameter ϵ). We consider three different values of ϵ—0.1, 1.0, and 10.0—that correspond roughly to high, medium, and low privacy levels. Recall that higher values are not necessarily unsafe but are theoretically easier to break.

Privacy principal The guarantees of differential privacy are for the records of the underlying data set. These records may or may not directly correspond to the higher-level privacy principal that the data owner wants to protect. Network data is interesting in that there are multiple possible privacy principals such as packets, flows, hosts, and services. If the underlying records are finer-grained than the intended principal (e.g., packets vs. hosts), no explicit guarantees are given for the principal.

Selecting an appropriate-granularity privacy principal is an important first step for the data owner. As a logistical matter, finer-grained records that share the same higher-level principal can be aggregated into one logical record using SQL-like views. Using this aggregated data will then provide guarantees as the level of the principal. But in general, the analysis fidelity will decrease as fewer records are able to contribute to the output statistics.

In this paper, we assume that the privacy principal is at the granularity of records in the dataset. This position is generous for analysis but it is also the starting point for beginning to understand the applicability of differential privacy in our context. If analysis noise is excessive even at this granularity, there is little hope. In the future, we intend to study the impact of using higher-level principals.

Packet-level analyses		Expressibility	High accuracy
Packet size and port dist.	(§5.1.1)	faithful	strong privacy
Worm fingerprinting [27]	(§5.1.2)	faithful	weak privacy
Flow-level analyses			
Common flow properties [30]	(§5.2.1)	could not isolate connections in a flow	strong privacy
Stepping stone detection [33]	(§5.2.2)	(one of the two) sliding windows were approximated	medium privacy
Graph-level analyses			
Anomaly detection [13]	(§5.3.1)	faithful	strong privacy
Passive topology mapping [9]	(§5.3.2)	used a simpler clustering method	weak privacy

Table 2: The analyses that we consider and summary results for them.

4. A PRIVATE ANALYSIS TOOLKIT

In this section we present a collection of tools that implement primitives that are common to many network trace analyses. The tools are applicable to data analysis broadly and represent the first practical implementations that are sensitive to privacy cost and added noise. We will arrive at specific networking analyses in the next section. Our toolkit and the associated analyses are publicly available [23].

4.1 The Cumulative Density Function

Often, in addition to simple aggregates like counts and averages, we are interested in understanding the underlying distribution itself, which may have informative ranges or modes. In networking analyses, distributions are often studied using the CDF: cdf(x) = number of the records with value $\leq x$. Measuring precise empirical CDFs with arbitrary resolution is not possible with differential privacy; as the resolution δ decreases, cdf(x) - cdf(x-δ) depends on only a few records in the data. We present three approaches to approximate the CDF.

A simple approach is to partition the range into buckets of a certain resolution and count, for each bucket, the records that fall in that bucket or a previous one. Let *buckets* be the set of values that represent the high end of each bucket, then:

```
foreach (var x in buckets)
  trace.Where(rec => rec.val < x).Count(epsilon);
```

This approach directly measures each value of cdf(x), but the standard deviation of the error is proportional to $|buckets|$.

A more advanced approach is to use the `Partition` operation to partition data into buckets:

```
tally = 0;
parts = trace.Partition(buckets, rec => rec.value)
foreach (var x in buckets)
  tally += parts[x].Count(epsilon);
  yield tally;
```

This approach has the advantage that the total privacy cost is independent of the number of buckets (i.e., resolution) but a limitation is that the error at each measurement accumulates to form the CDF. However, these errors cancel somewhat, and their standard deviation is proportional only to $\sqrt{|buckets|}$.

An even more advanced approach takes measurements at multiple resolutions and aggregates at most a logarithmic number of measurements to reproduce the full set of values for the CDF. A recursive function to implement this approach is:

```
CDF3(data, epsilon, max)
  if (max == 0)
    yield return data.Count(epsilon);
  else
    //--- partition data at max/2
    var parts = data.Partition(new int[] { 0, 1 },
                               x => x/(max/2));
    //--- emit counts for [0,max/2)
    foreach (var x in CDF3(parts[0], epsilon, max/2))
      yield return x;
    //--- a cumulative count for [0,max/2)
    var count = parts[0].Count(epsilon);
    //--- emit frequencies for [max/2, max)
    parts[1] = parts[1].Select(x => x - max/2);
    foreach (var x in CDF3(parts[1], epsilon, max/2))
      yield return x + count;
```

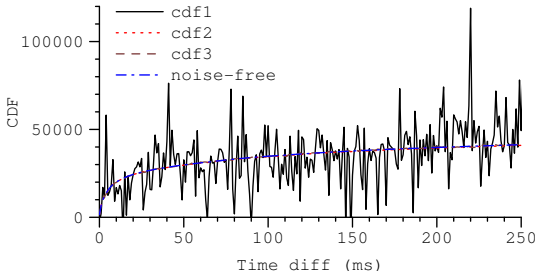
The standard deviation associated with each measurement is proportional to $\log(|buckets|)^{3/2}$.

Figure 1 compares these three approaches to the actual CDF for the time difference between a packet and its retransmission in the Hotspot trace. We discretize values to 1-ms granularity. The top graph shows that the error from the first approach is incredibly high, but the other two approaches are accurate. The bottom graph zooms in to show the distinction between the latter two approaches. We see that in the second approach yields a smoother estimate that mimics reality but consistently underestimates because the error accumulates across the range. (In a different run, we may see a consistent overestimation.) The errors with the third approach are generally lower but could represent an over- or under-estimation at individual points. In any case, with both these approaches the errors are relatively small and likely acceptable for most settings with modest absolute counts.

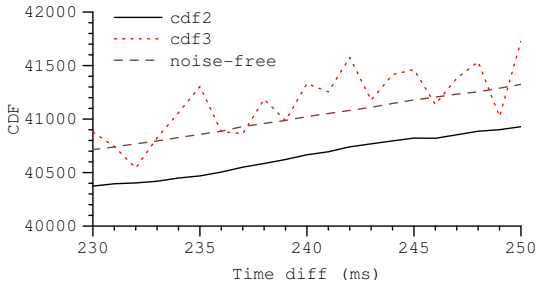
A natural consequence of noisy measurement is that the computed CDFs are not non-decreasing. If needed, the CDFs can be made non-decreasing through isotonic regression. Linear time algorithms (e.g., the "pool adjacent violators" algorithm [3]) can find the non-decreasing curve that minimizes the squared error distance from the input. Such smoothing can also increase the accuracy in some cases (e.g., cdf3 in Figure 1). But it is a non-reversible removal of information, so we do not do it by default.

4.2 Finding Frequent (Sub)strings

Many analyses need to identify substrings or values that occur frequently, for example, common payloads or addresses. While this may seem at odds with privacy, the presence or absence of individual records is not necessarily at risk; if a particular string occurs a large number of times, it is essen-



(a) Complete view of all three methods



(b) Zoomed in view of cdf2 and cdf3

Figure 1: Comparing three approaches for computing CDFs with the actual (noise-free) CDF. (a) The complete view shows that the first approach has high error but the other two are indistinguishable from the actual CDF. (b) The zoomed-in view shows the error behavior of the last two approaches.

tially a statistical trend that need not reveal the presence or absence of one of its representatives.

As a concrete example, consider the problem of learning the common strings of length B bytes. We might partition our set of packets by the all possible values, of which there are at most 256^B, and measure the number of records in each bin. Although the privacy cost is not high, the computational cost is exorbitant for even small values of B.

Instead, we can reveal common strings by asking about statistics of successive bytes. Initially, we partition the records into 256 bins based on the first byte, and count the number of records in each bin.

```
parts = data.Partition(bytes, rec => rec.str[0])
foreach (var byte in bytes)
  if (parts[byte].Count(epsilon) > threshold)
    yield return parts[byte]
```

All common strings contribute to the counters associated with their first bytes, which should be noticeably non-zero. Each byte with count greater than *threshold* can now be extended, by each of the 256 bytes, to form prefixes of length two. Again we can partition and count, using our new candidates and the first two bytes of each string, resulting in a set of viable prefixes of length two. This process continues until length B, at which point the counts correspond to the number of records with each distinct B byte string. We would have ideally culled most of the strings along the way, rather than at the very last step, as a monolithic partition operation would do. While we incur a higher privacy cost, due to the B rounds of interrogation, we can afford to take

string	true count	est. count	% err
2D2816FECDCAB780	3038504	3038500.005	-0.000
F389B84545A38BAF	92494	92505.050	0.012
E41903DCF7D86F2F	41600	41606.893	0.017
6F7E03DC833D6F2F	40279	40287.970	0.022
CD4F03DCE10E6F2F	40084	40087.437	0.009
B68503DCCA446F2F	37431	37448.584	0.047
58B403DC6C736F2F	36526	36537.877	0.033
41EA03DC55A96F2F	29625	29624.397	-0.002
9FBB03DCB37A6F2F	20715	20711.169	-0.018
7EEEB845D1088BAF	18976	18980.823	0.025

Table 4: True and noisy counts of the top-10 strings.

measurements with less accuracy as we face relatively fewer opportunities for false positives.

We used the procedure above to find the top 10 strings in the payloads of the Hotspot trace. Table 4 shows the hash value of the discovered strings, true and estimated counts, and the relative error. We see that the top 10 strings are discovered correctly, in order, and the error in the estimated count is low. The number 10 was an arbitrary choice for presentation; the computation produces counts for all strings whose counts exceed a user specified threshold with a user specified confidence.

4.3 Frequent Itemset Mining

A recurring theme in many data analyses is that commonly co-occurring items are a possible indication of correlation. The task of identifying frequently co-occurring items across an input list of item sets is called frequent itemset mining.

There are many algorithms for this task, including the popular `apriori` algorithm [1]. It starts with a collection of singleton sets and counts the number of times each occurs. Sets that have sufficient frequency are retained, and merged to form sets of size two, and so on.

Thus, the insight underlying this algorithm is similar to what we used for frequent substring counting. But a key difference from a privacy perspective is that the records, which are each essentially a set of items, must be partitioned amongst the candidate itemsets; each record can only contribute to the count for one candidate itemset even though it may support several. Consequently, if there are too many candidate itemsets it can be hard to assemble enough evidence for any one candidate.

We get over this hurdle by aggressively restricting the candidate item sets with high thresholds, focusing the support of the records and ensuring that we do not spread the counts too thin. Counter-intuitively, these high thresholds allow us to learn more. We omit implementation details for space constraints.

As one brief example of its use, we use it to discover the common sets of ports that are used simultaneously by hosts. Our discovered sets were very close to reality. The top-five, which are all correct, in the Hotspot trace are (22,80), (25,22), (443,80), (445,139), and (993,22).

5. NETWORK TRACE ANALYSES

We now survey our experiences at reproducing several analyses from the networking literature. We stress that while we consider a wide range of analyses, our experiences

may not be representative. Moreover, our reproductions are each only one of many possible ways of reproducing an analysis; different ways of measuring the same quantity may lead to different results.

Table 2 summarizes our findings. "Expressibility" reflects the faithfulness of our implementation to the original analysis, ignoring quantitative privacy constraints. That is, if the privacy allotment was arbitrarily high, would we reconstruct the original results or deviate from the specification of the original algorithm? To a first order, we find that we are able to reproduce the analyses, though some flexibility is required in reproducing the spirit of the analysis, if not the exact letter.

"High accuracy" indicates our qualitative assessment of what privacy level yielded highly accurate results; stronger privacy levels do not necessarily yield bad results (some do) but do produce noticeably different outputs. In all cases, medium privacy (ϵ= 1.0) produces admirable results. Picking an appropriate point in the privacy-accuracy trade-off requires a more concrete understanding of the data's sensitivity and the value of accuracy, but our results suggest several plausibly valuable locations on the privacy-accuracy curve.

5.1 Packet-level Analyses

We now present our results in more detail, beginning with packet-level analyses. Unless otherwise specified, we use the Hotspot trace.

5.1.1 *Packet-size and port distributions*

Two common packet-level analyses are measuring the distribution of packet sizes and ports. These are easy to reproduce with the CDF computation methods that we described earlier. We use the second method in our experiments.

Figure 2(a) shows the fidelity of the CDFs of packet length computed with the three values of ϵ that provide different privacy strengths. The graph also shows the real, noise-free CDF and error bars for each noisy CDF. We see that the error is minimal even at the strongest privacy level. As one measure of the overall accuracy, we compute the root mean square error (RMSE) as $\sqrt{\frac{1}{n}\Sigma_i(1-\frac{v_p[i]}{v_{nf}[i]})^2}$, where $v_p[i]$ and $v_{nf}[i]$ are the private and noise-free values at index i. At ϵ=0.1, the RMSE is only 0.01%.

This extremely low error implies that accurate results can be obtained even with far less data. Indeed, when we restrict our computations to only 1/10th of the data, the RMSE increases to only 0.02%.

We also see that privately computed CDFs correctly capture the interesting features of the distribution, for example, spikes at 40 and 1492 bytes. The former corresponds to TCP acknowledgments with no data, and the latter to the maximum packet size with IEEE 802.3 (which is used for wireless communication).

Figure 2(b) shows that similarly high-fidelity result are obtained for port distributions. At ϵ=0.1, the RMSE is only 0.07%. With 1/10th of the data, the RMSE is 0.7%. The error for ports is more than that for packet lengths because there are more unique ports, and thus there are in general fewer packets that contribute to port frequencies.

While packet length and port distributions may not seem the most exciting quantities, they are simply examples of CDFs of arbitrary packet statistics. Computations using more sensitive information (e.g., the CDF of the scores of a packet payload classifier) are similarly straightforward for an analyst to specify and to convince the data provider of the privacy guarantees.

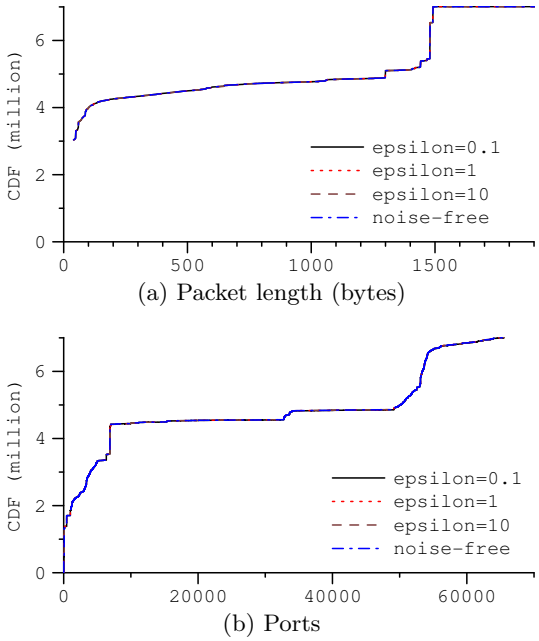


Figure 2: Packet length and port CDFs computed without noise and with different values of ϵ. The curves (and error bars) are all indistinguishable.

5.1.2 *Worm fingerprinting*

We now consider a more complex packet-level analysis which looks closely at packet payloads and depends critically on this very sensitive data. Automated worm fingerprinting [27] examines a stream of packets for frequently occurring payload substrings, with an additional "dispersion" requirement that the substring is originated by and destined to many distinct IP addresses.

The PINQ fragment grouping the packets by payload and restricting to those with the appropriate dispersion properties is:

```
trace.GroupBy(pkt => pkt.payload)
     .Where(grp => grp.Select(pkt => pkt.srcIP)
                      .Distinct()
                      .Count() > srcthreshold)
     .Where(grp => grp.Select(pkt => pkt.dstIP)
                      .Distinct()
                      .Count() > dstthreshold)
```

These packet groups are still hidden behind the privacy curtain, and while we could count the groups (2739 ± 10, with thresholds at 5), or consider other statistics thereof, we cannot (yet) directly view them.

To read out the interesting payloads, we leverage their frequency in the data set. We use the frequent string finding technique (§4.2) to spell out payloads that appear a significant number of times. This produces a list of candidate payloads, from which we want to evaluate each to see if it might be deemed suspicious. A simple PINQ fragment to produce the number of distinct destinations associated with each candidate payload is:

```
parts = trace.Partition(candidates, x => x.payload);
foreach (var candidate in candidates)
    parts[candidate].Select(x => x.dstIP)
                    .Distinct()
                    .Count(epsilon)
```

A similar fragment yields the distinct sources for each payload. The reported values for each payload are correct up to the error PINQ introduces.

With a dispersion threshold of 50 for sources and destinations, the noise-free computation yields 29 payloads. Searching for prefixes privately with ϵ values of 0.1, 1.0, and 10.0 reveals 7, 24, and 29 of these 29, respectively. That is, we miss 75%, 17% and 0% of the payloads. The missing payloads tend to correspond to payloads with low overall presence but above average dispersal.

Thus, unlike packet length and port analysis, the accuracy of worm fingerprinting is low at high privacy levels and high only at low privacy levels. Because the theoretical distribution of analysis error is known in advance, the analyst can judge that the results have low accuracy at high privacy levels.

The approach of [27] is extended in several ways in the paper. The extensions include reducing false positives by incorporating the destination port into the signature and sliding a window over the payloads to look for invariant content. We are able to express both these extensions in PINQ. But with them, we do not find any high-dispersal signatures in our trace, even in the absence of noise. Our monitored environment likely observes little worm activity because it is behind a single public IP address.

5.1.3 Summary

We showed results from two kinds of packet-level analyses at the opposite ends of the spectrum. One was simple distributions over packet sizes and ports, and the other was a more involved computation that considered payloads, ports, and IP addresses. We found that the both could be faithfully reproduced and the output fidelity was high at least at low privacy levels. Based on these results, we surmise that many other forms of packet-level analyses, such as various classification algorithms [10], can also be implemented in the differentially private manner.

5.2 Flow-level Analyses

We now investigate the feasibility of conducting flow-level analyses in a differentially private manner. These analyses differ from packet-level analyses as they consider properties across groups of packets. Rather than aggregate directly across packets, we need to first apply non-trivial computation across the packets to yield the derived statistics of interest.

5.2.1 Common flow statistics

A common operation for network analyses is to compute flow properties such as round trip time (RTT) and loss rate. To compute these statistics, we use the techniques used by Swing [30]. A flow refers to the standard 5-tuple.

Swing measures RTT of a flow by differencing the time between the TCP SYN and the following SYN-ACK. Considering only the handshake means that the results are not impacted by delayed acknowledgments. To reproduce these RTT values in PINQ, we join SYNs with SYN-ACKs, seeking pairs corresponding to common flows, with the ACK

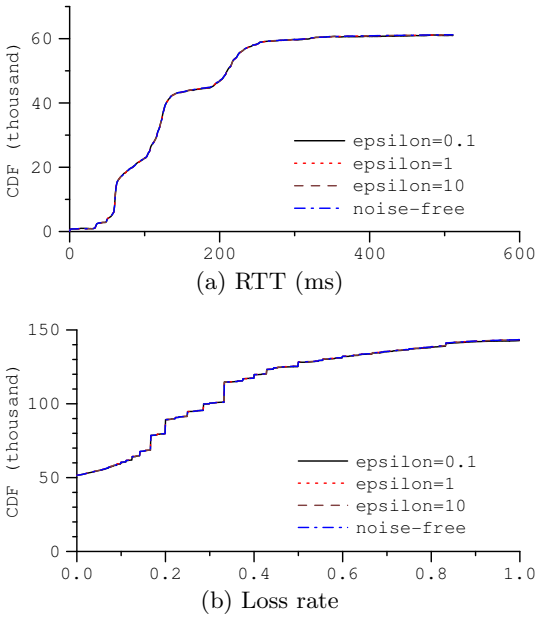


Figure 3: The CDF of RTT and loss rate computed without noise and with differential privacy. All curves (and error bars) are indistinguishable.

number of the second equal to the sequence number of the first, plus one.

```
syns = packets.Where(x => x.syn)
acks = packets.Where(x => x.syn && x.ack)
times = syns.Join(acks,
                  x => x.src + x.dst + (x.seqn + 1),
                  y => y.dst + y.src + y.ackn,
                  (x,y) => y.time - x.time);
```

Swing measures flow loss rate downstream of the monitored link using TCP retransmissions. When a packet is lost downstream, the monitor will observe a corresponding retransmission. We group packets by flow, and compare distinct sequence numbers to total packets:

```
trace.GroupBy(pkt => pkt.flow)
     .Select(grp => grp.Select(pkt => pkt.seq))
     .Select(grp => grp.Distinct().Count/grp.Count())
     .Select(x => 1.0 - x);
```

Once RTT and loss rates have been computed, we can study their distributions using the CDF primitive. Figure 3 shows the results for these two properties with the three privacy levels. RTT is computed only for flows for which we see both the SYN and its ACK. Loss rate is computed only for flows with more than 10 packets. We see that for both properties the results are high-fidelity even at the strongest privacy level. At ϵ=0.1, the RMSE for RTT is 2.8% and for loss rate is 0.2%.

We also considered other properties that Swing considers, including loss rate upstream of the monitor (computed using out-of-order packets) and path capacity (computed using the time difference and sizes of in-order packets). For these, the results are similar to those shown above.

There was one class of computations in Swing that we could not immediately reproduce in PINQ. This class operates at the level of connections, e.g., computing the number

of packets per connection. A (5-tuple) flow may include multiple TCP connections, and we could not isolate the connections within a flow using the currently available operations. This issue is not fundamental, however. The data owner could pre-process the traces to add a "connection id" field, or (as we are currently investigating) PINQ could be extended with more flexible grouping transformations. Once connections are identified, the connection-level analyses are straightforward.

5.2.2 Detecting stepping stones

We now consider an analysis that operates across packets of different flows rather than working within individual flows. This analysis detects stepping stone relationships between flows [33]. A stepping stone occurs when a computer is accessed indirectly, through a chain of one or more other computers. One scenario for such usage is to launch attack in a way that makes it it harder to trace back to the source.

Stepping stone detection [33] leverages the intuition that, for related interactive flows, the states of the flows are likely to go from idle to active together, many times. It establishes a time-out interval (T_{idle}=0.5 secs) after which a flow is considered idle, and a another time window (δ=40 ms) within which idle-to-active transitions of two flows are considered correlated. It then identifies as stepping stones pairs of flows that exhibit a high ratio of correlated idle-to-active transitions to all such transitions. To minimize false positives, it also constrains the correlated flows to occur in the same order multiple times and places a lower bound on the ratio of the ordered occurrences to idle-to-active transitions.

Identifying the set of idle-to-active transitions is a sliding window computation that we conduct in PINQ by bucketing time in buckets of width $2T_{idle}$. We group packets by a combination of flow and bucket. Each group can contain at most one activation in it's second half—the last—and we have enough context to confirm this packet as an activation or not.

```
packets.GroupBy(x => new {x.flow,x.time/(2*T_idle)})
       .Where(/* if last packet is an activation */)
       .Select(x => x.Last())
```

This captures roughly half of the activations. To produce the remaining we shift each time by T_{idle} and apply the same operation, moving packets from the front half of each bucket to the rear.

Next, we need to identify correlated activations across flows. While we could reproduce the sliding window in the same manner as above, the double groupings required double the noise we must suffer. We find that a better option is to bin the activations by time, and then run frequent itemset mining to identify pairs of flows that are frequently activated together. This trade-off between fidelity to the source algorithm and privacy efficiency is one we will see again. Designing analyses for privacy from the ground up is likely to yield better results in settings where privacy is mandatory. The pseudo code for binning flows by time is:

```
activations.GroupBy(x => x.time / delta)
           .Select(x => x.Select(y => y.flow)
                         .Distinct())
```

Finally, we need to evaluate if the pairs thus produced are stepping stones by the original criteria. To evaluate a given candidate pair, we simply count the number of bins containing both. To evaluate many pairs, we first `Partition` the activations by flow, which reduces the privacy cost dramatically.

ϵ	noisy corr.	noise-free corr	false positives
0.1	0.06 ± 0.07	0.03 ± 0.01	18/20
1.0	0.72 ± 0.10	0.76 ± 0.12	1/20
10.0	0.78 ± 0.03	0.82 ± 0.05	2/20

Table 5: Evaluating private detection of stepping stones.

In the Hotspot trace, we find a surprising number of correlated flows (even with non-private computations), likely because of the couplings between flows introduced by the wireless channel. This likely suggests that the original stepping stone algorithm needs to be recalibrated for wireless traffic. For us, however, this complicates the task of frequent itemset mining as the data becomes too dense. We could tweak T_{idle} and δ, but that makes validation harder.

Instead, to reduce density and being able to compare against the original parameters, we focus on the set of flows with $[1200, 1400]$ activations. We compare against a faithful implementation (in Perl) that does not approximate the task of identifying correlated flows. Thus, the comparison includes errors introduced by privacy constraints as well as algorithmic approximation.

Table 5 shows the results. For each value of ϵ, it shows the average and standard deviation of the approximate correlation for the top-twenty flows pairs, approximately computed with bucketed correlations. For those flow pairs, it also shows the actual average correlation scores (computed with the Perl script), and what fraction had no actual correlation. We see that ϵ=0.1 has a very high false positive rate. But higher values of ϵ have good accuracy, suggesting that stepping stones can be detected accurately with "medium" privacy levels. That we see accurate results at these privacy levels also indicates that the impact of algorithmic approximation is low. The threshold for correlation used in the original analysis was 0.3, and every non-false positive candidate for ϵ at 1.0 and 10.0 was above this threshold.

5.2.3 Summary

We presented two kinds of flow-level analyses. One computes statistics within flows and another that is based on correlations among different flows. Though there are rough edges (that are resolvable), in both cases, we are able to capture the essence of the analysis and the output is high fidelity. Based on these, we believe that many other forms of flow-level analyses can be conducted in a differentially private manner. For instance, we are able to reproduce the association-rule mining based analysis of Kandula et al. [12] with a high fidelity; we omit results due to space constraints.

5.3 Graph-level Analyses

We now turn our attention to analyses that focus on network-wide properties rather than those of individual packets or flows. As with the previous two sections, some statistical properties are relatively easy to produce: distributions of in and out degrees of nodes in the graph, restricted to various ports or protocols, distributional properties of computed quantities of edges (e.g., the distribution of loss rates across edges in the graph). Some useful properties, such as the diameter of the graph or the maximum degree, are difficult or impossible to compute because they rely on a handful of

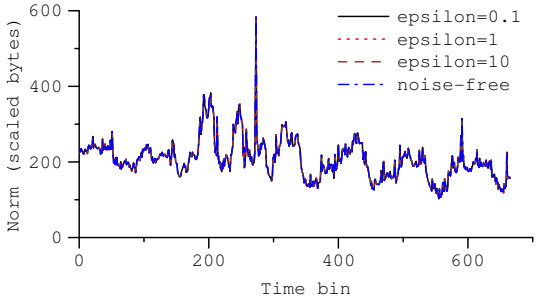


Figure 4: The norm of anomalous traffic computed with and without privacy. All four lines are indistinguishable.

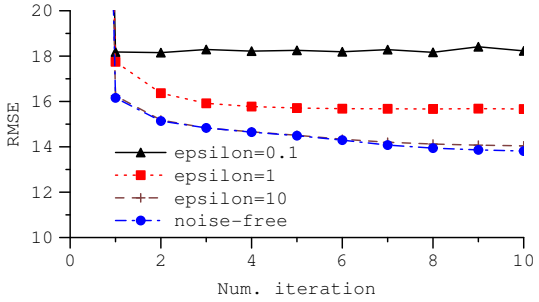


Figure 5: Clustering error with and without privacy.

records. We consider two complex graph-level analyses that lie between these two extremes.

5.3.1 Anomaly detection

The first graph-level analysis that we consider is the detection of network-wide traffic anomalies by observing link-level traffic volumes across time. We follow the analysis proposed by Lakhina et al. [13]. They first assemble a matrix indexed by link and time bucket, where each entry corresponds to load on the link at that time. They then apply principal components analysis (PCA) to this matrix and use the first few factors to represent "normal" traffic. Entries not well described by these factors represent substantial deviations from the normal, and they are labeled as anomalies and flagged for inspection.

While the algorithm is mathematically sophisticated, we will have little trouble adapting the approach to work within PINQ. The first step, computing the link×time load matrix is an aggregation:

```
rows = trace.Partition(links, x => x.link)
foreach (var link in links)
  vals = rows[link].Partition(times, x => x.time);
  foreach (var time in times)
    vals[time].Count(epsilon);
```

While the counts are noisy, the definition of a volume anomaly is robust to small counting errors, and no significant anomaly should go unnoticed. This robustness can be seen in Figure 4 even at the highest privacy level. The graph shows, for the IspTraffic dataset, the volume of anomalous traffic, i.e., bytes that are badly explained by the first few singular vectors of the traffic matrix. Despite the complexity of the analysis, the relatively low volume of noise added to each measurement and the robustness of the technique lead to results that are indistinguishable from the noise-free version. The anomalies in the network, e.g., at time unit of 270, clearly stand out. The RMSE at ϵ=0.1 with respect to noise-free results is 0.17%.

5.3.2 Passive network discovery

Eriksson et al. propose a novel approach to map network topology [9]. It takes as input a collection of hop count measurements between a large number of IP addresses and a few monitors. It infers network topology by clustering IP addresses based on these measurements—two IP addresses that have similar hop counts to most of the monitors are likely topologically close. This work follows the clustering with small number of active measurements to each of the identified clusters.

We focus on whether we can reproduce the clustering analysis in a private manner; active measurement require non-private information by necessity. This separation is not uncommon in privacy-preserving data analysis: a large volume of protected data is analyzed to find trends, after which a smaller amount of privileged data is subjected to arbitrary computation involving the learned trends.

The clustering analysis starts by establishing the average value of each monitor across all IP addresses, to be used in lieu of absent readings.

```
average = monitor.Average(epsilon, x => x.hops);
```

The monitors are then assembled into a collection of vectors, one for each IP address, and one coordinate per monitor. Addresses not observed at a monitor result in the average value for that coordinate:

```
monitors.Aggregate((x,y) => x.Concat(y))
        .GroupBy(x => x.IP)
        .Select( /* additional logic */)
```

So assembled, the set of vectors can be subjected to standard clustering algorithms. We use k-means clustering of PINQ. The original analysis uses Gaussian EM instead, an extension of k-means using covariance matrices for each cluster. While Gaussian EM is also expressible, it has a higher privacy cost and is consequently less accurate for us. This calls into light the trade-off between algorithmic complexity and accuracy; more complex algorithms can give better results in the absence of privacy constraints, but if their sophistication requires looking "too closely" at the data, the necessary noise to preserve privacy can counteract these gains.

The data used by Eriksson et al. is hop count (inferred using TTL) from scanning IP addresses to honeypot monitors. We run our analysis on the IPscatter dataset, which is similar. It has hop count measurements from PlanetLab nodes (as monitors) to large number of IP addresses.

Figure 5 shows the results with different values of ϵ as well as without privacy. It plots the objective function of the k-means optimization, the average distance from a point to its nearest cluster center, against the number of iterations conducted. Nine centers are used, initialized to a common random set of vectors for each execution. For each value of ϵ, each iteration of the algorithm consumes another multiple of the privacy cost. After 10 iterations, a value of ϵ=0.1 costs 1. However, given the flatness of the curves, for a fixed

privacy budget, the appropriate strategy may not be to run ten iterations at one-tenth the accuracy.

The curves reveal that at the strongest privacy level (ϵ=0.1) the RMSE is worse by 50%. The medium privacy level is much closer and its error may be acceptable. (The implications of variation in cluster quality on the reconstructed network topology is beyond the scope of this paper.) The weakest privacy level, however, is able to provide results almost identical to the non-private computation.

5.3.3 Summary

We considered two graph-level analyses. We were able to reproduce the anomaly-detection analysis faithfully because most of its complex computations are on heavily aggregated data that is less hindered by privacy constraints. The passive network discovery analysis yielded high-fidelity results only with weak privacy guarantees. It also exposed a trade-off between algorithmic complexity and privacy cost. Given that these two analyses are fairly involved, our experience suggests that many other graph-level analyses can be conducted in a differentially private manner.

6. RELATED WORK

The dominant method for data sharing today is trace anonymization [16, 31, 22]. However, many researchers have shown that anonymization is vulnerable to attacks that can extract sensitive information from the traces [5, 26, 31, 21]. The utility of anonymized traces is further limited by the removal of sensitive fields, critical for certain analyses [21].

Because of these shortcomings of anonymization, researchers have begun exploring mediated trace analysis. There are three proposals to our knowledge, none of which match the strong and direct privacy guarantees of differential privacy. First, Mogul and Arlitt's SC2D relies on the use of pre-approved analysis modules and human verification to preserve privacy [19]. Given the complexity and diversity of network analyses, it is unclear if human verification is practical and what guarantees it can provide.

Second, Mirkovic's secure queries [17] are conceptually similar to our work in that the analysis is expressed in a high-level language and the analysis server is tasked with ensuring privacy. The privacy requirements are inspired by differential privacy as well. However, privacy is enforced using a set of ad hoc rules whose eventual properties are poorly understood. Further, while we show that a range of analyses can be accurately done using our methods, Mirkovic does not evaluate the usefulness of secure queries.

Third, Mittal et al. develop a method for quantifying the amount of information revealed by an analysis and propose that data owners refuse to support analyses that leak more than a threshold [18]. While intriguing, this approach is vulnerable to targeted attacks. As previously discussed, single bits can be arbitrarily sensitive, and the refusal reveals a bit in itself. Differential privacy reveals less than a bit about each record, but many bits about aggregate statistics.

Differential privacy is a recent concept and its practical utility is an open question that can be answered only by applying it to several domains. Along with McSherry and Mironov, who study Netflix recommendations [15], and Rastogi and Nath, who study distributed time-series [24], our work helps to further an understanding of this question. Reed et al [25] recently proposed an analysis language similar to PINQ to detect botnets in a differentially private manner. While this approach has not been evaluated yet, our experience suggests that it can be effective.

7. DISCUSSION AND OPEN ISSUES

Our results indicate that differential privacy has the potential to be the basis for mediated trace analysis, which will enable data owners to let other analysts extract statistical information in a provably private manner. The limitations of differential privacy with respect to output fidelity and the need to implement the analysis in a high-level language are surmountable for a large class of analyses.

Retrospectively, the success of differential privacy in this domain stems from two factors. First, many analyses seek aggregate statistical trends and common patterns in the data. For such analyses, individual records contribute only a small fraction to each output value, which implies that only a small amount of noise can guarantee privacy. Second, the computations that many analyses conduct directly over individual records are rather simple and thus easy to express. Any complicated computations (e.g., clustering, PCA) are conducted only over aggregate data that can first be extracted privately with a high fidelity. These properties may not hold for all analyses but they appear to hold for large class of analyses.

We do not claim that implementing analysis in a differentially private manner is straightforward. We ran into many challenges and counter-intuitive behaviors. Some analyses yielded low fidelity results at strong privacy levels; high output fidelity could be achieved only at weak privacy levels. Between, at medium privacy settings, the accuracy was rarely bad, but distinguishable from the truth. Whether this is sufficient depends on the needs of the analyst, and the available privacy resources. As more thought is put into algorithm (re-)design, we expect these trade-offs to improve.

Some computations that are easy otherwise (e.g., sliding windows) can have a high privacy cost. Others, such as empirical CDFs with arbitrary resolutions, are fundamentally impossible to do in a differentially private manner. Further, there are multiple ways to implement the same analysis, some more privacy efficient than others. A worthwhile task for the future is to educate networking researchers on the concept of privacy efficiency, which is distinct from, and sometimes counter to, the more familiar concept of computational efficiency. A related one is to develop a library with privacy-efficient implementations of common primitives used by networking analyses. The toolkit presented in this paper is a first step in that direction [23].[2]

This paper is by no means the final word on the use of differential privacy for mediated trace analysis; there are several policy-related and practical challenges that must first be fully explored. One such challenge, which we mentioned in §3, is developing support for coarser-granularity privacy principals (e.g., flow or hosts) even when the underlying data is at a finer-granularity (e.g., packets).

Another challenge is developing guidelines for data owners on what privacy level (parameter ϵ) to set for their datasets.

[2]Expressing analyses in high-level languages makes them easier to debug and maintain as well. In the specific context of PINQ, because it is based on LINQ, the analyses will also automatically scale to a cluster [32]. Today, for flexibility, most networking analyses are written in low-level languages (e.g., C, Perl). Our survey provides evidence that the community can afford to move to high-level languages.

While we explore a range of levels in our work, owners will have to decide on specific levels to us for their data. There is unlikely to be a single answer for all situations. Instead, the appropriate level should be based on a combination of data sensitivity, the value of the analysis, acceptable noise-level, and the trust in the analyst.

Yet another challenge is managing the impact of repeated use of the same data, by the same analyst or by different analysts. Each use leaks some private information (in theory) and successive uses leak more information. Differential privacy provides useful guidance on this issue. Two analyses with privacy cost $c1$ and $c2$ have a total privacy cost at most $c1 + c2$. Using this property, the data owners can enforce various policies such as limiting the total privacy cost per analyst or across all analysts. They can also reduce privacy cost (i.e., increase ϵ) with time such that the data is available longer but the added noise increases with time.

Resolving these challenges requires balancing usability and privacy. With the strong foundation provided by differential privacy, we are optimistic that they can be resolved to the satisfaction of many data owners.

Acknowledgments We are grateful to Saikat Guha, Suman Nath, Alec Wolman, the anonymous reviewers and our shepherd, Walter Willinger, for feedback on this paper. We also thank Stefan Savage for suggesting the worm fingerprinting and stepping s analyses early in the project.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[2] AOL search data scandal. `http://en.wikipedia.org/wiki/AOL_search_data_scandal`. Retrieved 2010-16-01.

[3] M. Ayer, H. Brunk, G. Ewing, W. Reid, and E. Silverman. An empirical distribution function for sampling with incomplete information. *The Annals of Mathematical Statistics*, 26(4), 1955.

[4] R. Chandra, R. Mahajan, V. Padmanabhan, and M. Zhang. CRAWDAD data set microsoft/osdi2006 (v. 2007-05-23).

[5] S. E. Coull, C. V. Wright, F. Monrose, M. P. Collins, and M. K. Reiter. Playing devilćs advocate: Inferring sensitive information from anonymized network traces. In *NDSS*, 2007.

[6] CRAWDAD: A community resource for archiving wireless data at Dartmouth. `http://crawdad.cs.dartmouth.edu/`.

[7] C. Dwork. Differential privacy. In *ICALP*, 2006.

[8] C. Dwork, F. Mcsherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, 2006.

[9] B. Eriksson, P. Barford, and R. Nowak. Network discovery from passive measurements. In *SIGCOMM*, 2008.

[10] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2), 2001.

[11] The Internet traffic archive. `http://ita.ee.lbl.gov/`.

[12] S. Kandula, R. Chandra, and D. Katabi. What's going on? Learning communication rules in edge networks. In *SIGCOMM*, 2008.

[13] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM*, 2004.

[14] F. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *SIGMOD*, 2009.

[15] F. McSherry and I. Mironov. Differentially private recommender systems: building privacy into the Netflix prize contenders. In *KDD*, 2009.

[16] G. Minshall. tcpdriv. `http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html`.

[17] J. Mirkovic. Privacy-safe network trace sharing via secure queries. In *workshop on Network Data Anonymization*, 2008.

[18] P. Mittal, V. Paxson, R. Summer, and M. Winterrowd. Securing mediated trace access using black-box permutation analysis. In *HotNets*, 2009.

[19] J. C. Mogul and M. F. Arlitt. SC2D: An alternative to trace anonymization. In *MineNet workshop*, 2006.

[20] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy*, 2008.

[21] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. *SIGCOMM CCR*, 36(1), 2006.

[22] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *SIGCOMM*, 2003.

[23] Network trace analysis using PINQ. `http://research.microsoft.com/pinq/networking.aspx`.

[24] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD*, 2010.

[25] J. Reed, A. J. Aviv, D. Wagner, A. Haeberlen, B. C. Pierce, and J. M. Smith. Differential privacy for collaborative security. In *EuroSec*, 2010.

[26] B. Ribeiro, W. Chen, G. Miklau, and D. Towsley. Analyzing privacy in enterprise packet trace anonymization. In *NDSS*, 2008.

[27] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *OSDI*, 2004.

[28] N. Spring, R. Mahajan, and T. Anderson. Quantifying the causes of path inflation. In *SIGCOMM*, 2003.

[29] L. Sweeney. k-anonymity: A model for protecting privacy. *Int'l Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems*, 10(5), 2002.

[30] K. V. Vishwanath and A. Vahdat. Swing: realistic and responsive network traffic generation. *ToN*, 17(3), 2009.

[31] J. Xu, J. Fan, M. Ammar, and S. Moon. Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *ICNP*, 2002.

[32] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Úlfar Erlingsson, P. K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, 2008.

[33] Y. Zhang and V. Paxson. Detecting stepping stones. In *USENIX Security*, 2000.