A Framework for Safely Publishing Communication Traces

Abhinav Parate and Gerome Miklau
University of Massachusetts, Amherst
Department of Computer Science
140 Governors Drive, Amherst, MA
aparate@cs.umass.edu, miklau@cs.umass.edu

Technical Report No. 2009-040

ABSTRACT

A communication trace is a detailed record of the communication between two entities. Communication traces are vital for research in computer networks and protocols in many domains, but their release is severely constrained by privacy and security concerns.

In this paper, we propose a framework in which a trace owner can match an anonymizing transformation with the requirements of analysts. The trace owner can release multiple transformed traces, each customized to an analyst's needs, or a single transformation satisfying all requirements. The framework enables formal reasoning about anonymization policies, for example to verify that a given trace has utility for the analyst, or to obtain the most secure anonymization for the desired level of utility. Because communication traces are typically very large, we also provide techniques that allow efficient application of transformations using relational database systems.

1. INTRODUCTION

A *communication trace* is a detailed record of the communication between two entities. Communication traces arise in a variety of settings and include network traces, phone toll records, instant-messaging transcripts, among others. Each record in a communication trace typically identifies a source and a destination, along with descriptive fields such as time stamp, transmitted content, length of transmission, and communication ports.

Communication traces are vital to research into traffic analysis, communication protocols, routing in networks, and security of communication networks. Unfortunately the public release of communication traces remains highly constrained by privacy and security concerns and the lack of available traces is a serious concern for researchers [5, 16].

Clearly the content of the communication (e.g. the packet content in an IP trace, or message content in an IM trace) is too sensitive to release – it is typically not the focus of analysis and is omitted immediately from analyzed traces. But even without transmitted content, a communication log includes the identity of the source and destination, timing information, and other sensitive communication details. Traces may therefore contain highly sensitive information about users of a communication protocol, about policies or organizational structure of an institution, etc.

The safe release of communication traces is a significant challenge. First, communication traces are transactional in nature, with information about entities spread across multiple records, and correlations between records. Conventional k-anonymization [15, 17] of a communication trace will tend to destroy these features. Recent variants of k-anonymization for transactional data have been proposed [18], but are focused on conceptions of utility that are inappropriate for communication traces (such as mining association rules). Differentially private mechanisms can be used to safely publish statistics about a communication trace, but statistics are insufficient for most communication trace analysis. The second challenge to protecting communication traces is their massive size. For example, a trace recording internet protocol (IP) packets at a medium size institution (used in later experiments) can produce 7.5GB traces every hour. Many proposed anonymization schemes simply cannot scale to such large data sets.

Our approach In this work we propose an approach to communication trace publication emphasizing utility and scalability. We address the problem faced by a *trace owner* who wishes to allow a group of independent *analysts* to safely study a communication trace.

The first component of our approach is a set of simple, formally-defined *transformation operators* that are applied to the trace to remove or obscure information. These include encryption, field removal, and domain translation. Transformation operators can be combined to form composite transformations. The output of the chosen composite transformation is published, and can be thought of as a safe *view* of the original trace.

Unlike most approaches to trace anonymization (in which the trace owner generates a single anonymized trace) we provide a framework for the trace owner to anonymize a trace for the needs of a particular analysis, releasing multiple traces. The published traces can be more secure because they provide only the needed information, omitting everything else. Our publication framework is illustrated informally in Figure 1. The figure shows an original trace T transformed in four different ways, for use by different analysts. Trace T_1 contains sufficient information for both analysts A and B. Trace T_2 is devised for use exclusively by the analyst C, and trace T_3 is customized for the needs of analyst D. An alternative to publishing both trace T_2 and T_3 is to derive the single trace T_{23} which can support analysts C and D simultaneously.

The second component of our approach is input from the analyst. We assume the requesting trace analyst provides a description of the information needed for analysis. We propose a simple language for *utility constraints* which express the need for certain relationships to hold between the original trace and the published

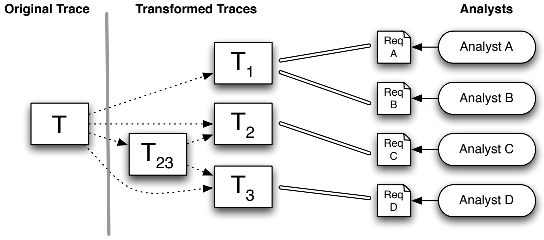


Figure 1: The proposed trace protection framework: the original trace T may be transformed in multiple ways (T_1, T_2, T_3, T_{23}) to support the requirements of different analysts.

trace. The constraints can require, for example, that certain fields are present and unmodified, or that other values can be obscured as long as they preserve the ordering relationship present in the original trace. It is usually straightforward to determine the constraints that are needed to support a particular analysis; we provide an example of a real traces analysis and its corresponding utility requirements in Section 2.

The third component of our approach is the formal evaluation of privacy and utility. Because both the transformations and the utility requirements of analysts are specified formally, it is possible for the trace owner to analyze trace publication scenarios precisely. In particular, the trace owner can: (1) decide whether a composite trace transformation satisfies an analyst's requirements, implying that the desired analysis can be carried out on the transformed trace with perfect utility; (2) compute the most secure transform satisfying a given set of analyst requirements; (3) compare the security of transforms or analyze the impact of a collusion attack that might allow published traces to be combined; and (4) find the single transform that satisfies the requirements of multiple analyses.

The result of our contributions is a framework in which basic trace transformation operations can be applied efficiently, and with a precise, formal understanding of their impact on trace utility and privacy.

Distinguishing our approach Our approach to protecting communication traces is distinguished in related two ways: we emphasize utility as the primary goal of trace transformation, and we publish multiple customized traces.

Most anonymization techniques impose a hard privacy condition and offer the best utility possible while satisfying this condition. As described above, for communication traces the utility is usually unsatisfactory with existing approaches. We instead impose a hard utility condition and maximize privacy with respect to it. This is an unconventional approach, but one that we believe is justified by the lack of acceptable tools for managing communication traces and the great societal value of sharing communication traces. There is some evidence that others are adopting a perspective emphasizing utility. The PREDICT repository [1] is a government-funded project which collects and releases traces. Although basic anonymization is performed, the released traces must be useful to analysts, and authentication of the requesting analysts is adopted as an additional protection mechanism.

Further, as illustrated in Figure 1 we provide utility and security by customizing the trace transformation to the particular needs of an analyst. This approach is motivated by an informal survey of recent published research on trace data sets (and anecdotal evidence from trace researchers themselves) that suggests that many individual studies could be performed on customized versions of the traces that would appear to be substantially safer than those published under conventional anonymizing transformations.

Admittedly, the trace owner in our framework must make more fine-grained choices about which transformed traces to publish to which users, and must compute and publish multiple anonymized traces instead of just one. We believe that the benefits to trace security warrant this added effort. Our transformation operators are efficient to apply, and we provide a number of tools to help the trace owner make publication decisions. In addition, the trace owner can always choose to publish a single trace supporting multiple analysts. For example, in Figure 1, publishing trace T_{23} may be easier than publishing both T_2 and T_3, but may require a sacrifice in privacy as, intuitively, Analyst C will receive some additional information used in the trace analysis D.

An additional concern with publishing multiple traces are attacks in which a single party poses as two or more analysts, or two or more analysts collude. In this case, different published views of the trace could be combined to reveal more information than intended. In the absence of a trustworthy authority validating the identities, it is a challenge to counter such attacks. We can however analyze the risk of collusion formally: in Section 4 we show that it is possible to bound how much a group of colluding parties can learn, distinguishing between cases where collusion is a serious risk and cases where little can be gained from collusion.

The remainder of the paper is organized as follows. In Section 2, we apply the main components of our framework in an example scenario. Section 3 describes our trace transformation operators and our language for specifying analyst requirements. Section 4 describes the formal analysis of transformations. Section 5 measures the privacy of sample transformations quantitatively through experiments on a real network trace, and Section 6 proposes techniques for fast trace transformation in a database system.

2. EXAMPLE: INFERRING TCP CONNECTION CHARACTERISTICS

Throughout the paper we use IP-level packet traces as a running example. Such traces (see Table 2(a) for illustration) are widely studied by the networking community and exemplify many of the challenges of working with communication traces. Our publication framework is general, however, and can be used for communication traces from many domains.

In this section, we provide an overview of our framework by describing an example of a real study from the area of network research as carried out by Jaiswal et al. [8]. We derive the basic requirements that must be satisfied by any usable trace, we describe an anonymizing transformation, and finally, we verify the transformation satisfies the requirements and assess the privacy of the trace.

Analysis Description

A TCP connection is identified by two IP addresses $(ip1, ip2)$ and two ports $(pt1, pt2)$, corresponding to the sender and receiver. Jaiswal et al. study the characteristics of TCP connections through passive monitoring [8]. Their study focuses on estimating the sender's congestion window and the round-trip time (*RTT*) taken by a network packet in the connection. In this analysis, the congestion window is estimated using a finite state machine(FSM). All the values required for FSM are present in the IP traces. The estimation of variable, RTT, is done by computing the time differences between various occurrences of a packet at the observation point of packets. The full details of this estimation can be seen in [8].

Utility Requirements

Table 1: *A semi-formal description of utility requirements sufficient to support the example analysis of TCP connection properties.*

Semi-Formal Utility Requirements	
Any tuple t in trace	Any tuples $t1, t2$ belonging to same connection in trace
`PRESERVE` $(t.syn)$ `PRESERVE` $(t.ack)$ `PRESERVE` $(t.window)$	`PRESERVE` $(t1.seq\_no \leq t2.seq\_no)$ `PRESERVE` $(t1.seq\_no - t2.seq\_no)$ `PRESERVE` $(t1.ts \leq t2.ts)$ `PRESERVE` $(t1.ts - t2.ts)$ `PRESERVE` $(t1.seq\_no == t2.ack\_no)$
where `PRESERVE` $(expr) \equiv expr_T = expr_{\phi(T)}$ i.e. value of $expr$ evaluated over tuples in trace T must be equal to value when $expr$ is evaluated over transformed tuples in $\phi(T)$	

Based on this description, we present the requirements or the sufficient conditions that must be satisfied by any transformed trace supporting the analysis described.

1. **R1** The trace must include the $type$ of the packet (SYN or ACK) and actual value of the $window$ field.
2. **R2** The trace must allow the analyst to order the same-connection packets by sequence numbers(seq_no) or timestamps(ts).
3. **R3** The trace should preserve the relative difference for values in ts and seq_no fields for the same-connection packets.
4. **R4** The seq_nos of the sender's packets and the acknowledgement numbers(*ack_no*) of receiver's packets of same connection should be comparable for equality in the trace.

The above requirements are specified semi-formally as a set of constraints given in Table 1. The formal requirements are constraints stating that certain relationships must hold between the original trace and the anonymized trace. The full description of our specification language is given in Section 3.2.

Trace Anonymization
Next we describe a simple transformation that provably satisfies the above utility requirements. For this transformation, we concatenate the fields $(ip1, ip2)$, encrypting the concatenated string to obtain anonymized $(ip1, ip2)$ fields. Similarly, we obtain anonymized $(pt1, pt2)$ by concatenating $(pt1, pt2, ip1, ip2)$ and encrypting it. This will map same $(pt1, pt2)$ values to different values if they are from different connections. The fields *ts* is anonymized by linear translation such that the minimum value in it for each group of records identified by $(ip1, ip2, pt1, pt2)$ becomes 0. For example, if the values of *ts* were 150,165 and 170, it will be linearly translated to 0,15,20. The field *seq_no, ack_no* are anonymized similarly such that the values are translated by the same amount in both the fields. We do not anonymize any other field, but we remove any field which is not required in the analysis.

In Section 3.1, we provide a basic set of formal transformation operators. The anonymization scheme described above can be expressed by composing these operators. This composite transformation function ϕ is given by:

$$\begin{aligned}\phi \;=\; & \Pi_X \circ E_{\{ip1,ip2\},\kappa_1} \circ E_{\{pt1,pt2\}(ip1,ip2),\kappa_2} \circ T_{\{ts\}(C)} \\ & \circ T_{\{seq\_no,ack\_no\}(C)} \circ I_{\{dir,window,syn,ack\}}\end{aligned}$$

Here $C = \{ip1, ip2, pt1, pt2\}$, E is encryption operator, T is translation operator, Π is projection operator, I is identity operator, X is the set of required attributes and κ_1, κ_2 are keys for encryption function.

As an example, the records in the sample trace given in Table 2(a) are transformed using above transformation function ϕ, to obtain the anonymized view given in Table 2(b). The encrypted values have been replaced by variables for clarity.

Provable Utility
The utility analysis verifies that the anonymization scheme, defined by the composite transformation function, satisfies the constraints.

Informally, as we do not anonymize *syn*, ack and $window$ fields in the trace, the type information of the packet and actual $window$ value is available, satisfying *R1*. The encryption of connection fields still supports grouping together of records in same connection (*R2*). The linear translation of *ts* and *seq_no* preserves the relative order (*R2*) and the relative differences in these fields (*R3*). By using the same transformation for *seq_no* and *ack_no*, we make sure that *R4* is satisfied, allowing equality tests on these fields.

The formal verification process requires the formal description of requirements and the anonymization scheme and it has been described in detail in section 4.1.

Privacy Analysis
Our publication mechanism can be seen as an enforcement tool, but the choice of which transformations of a trace to publish to which users is an important policy decision. The trace owner must make policy decisions based on the benefit of releasing the trace and the potential risk of disclosures.

To aid in this decision, we describe both *static* and *dynamic* disclosure analysis. The static analysis of transformations holds for all input traces, and allows for the relative comparison of transformations based on their information content. We show in Section 4 how to derive conclusions like ϕ_2 is more secure than ϕ_1, which enables general policies such as "a trace transformed under a transformation at least as secure as ϕ_1 can be released to any internal collaborator". The trace owner can also deduce the most secure transform that satisfies a given set of utility constraints.

The advantage of conclusions from static analysis is that they hold for all traces. But it is sometimes necessary to assess aspects of disclosure particular to an individual trace. In Section 5 we perform dynamic analysis of trace disclosure by simulating specific re-identification attacks on a trace, and by measuring the likelihood of accurating combining two transformations of a trace in a collusion attack.

3. TRANSFORMATION FRAMEWORK

In this section we describe the two main objects of our framework: *operators*, used by the trace owner to define trace transformations, and *constraints*, used by analysts to express utility requirements.

3.1 Trace transformation operators

The following transformation operators are applied to a trace in order to obscure, remove, or translate field values. Each transformation operator removes information from the trace, making it more difficult for an adversary to attack, but also less useful for analysts. The trace owner may combine individual operators to form composite transformations, balancing utility and security considerations. The output of a composite transformation will be released to the analyst.

Operator descriptions

We consider a communication trace as a table consisting of *records*. Each record consists of fixed number of fields and a timestamp field.

Projection The simplest operator is *projection*, which is similar to relational projection operator. Projection is denoted Π_X for retained attributes in X. The duplicates are retained by the operator.

Table 2: (a) Example of IP-level trace (b) Trace transformed under ϕ as described in Section 2

(a)

ts	*ver*	*ip1*	*ip2*	*pt1*	*pt2*	*dir*	*seq_no*	*ack_no*	*window*	*syn*	*ack*
30	4	172.31.1.34	172.31.2.212	22	22	→	5000	7280	8760	0	1
31	4	172.31.1.34	172.31.2.212	22	22	→	5012	7280	8760	0	1
32	4	172.31.1.34	172.31.2.212	22	22	←	7280	5024	65110	0	1
32	4	172.31.1.34	172.31.2.212	22	22	→	5024	7280	8760	0	1
31	4	172.31.1.34	172.31.2.212	80	9080	→	4780	8214	6432	0	1
30	4	172.31.1.34	172.31.2.89	80	9080	→	1000	1280	17424	0	1
31	4	172.31.1.34	172.31.2.89	80	9080	→	1012	1280	17424	0	1
32	4	172.31.1.34	172.31.2.89	80	9080	→	1024	1280	17424	0	1

(b)

ts	*ip1* *ip2*	*pt1* *pt2*	*dir*	*seq_no*	*ack_no*	*window*	*syn*	*ack*
0	c_1	p_1	→	0	2280	8760	0	1
1	c_1	p_1	→	12	2280	8760	0	1
2	c_1	p_1	←	2280	24	65110	0	1
2	c_1	p_1	→	24	2280	8760	0	1
0	c_1	p_1'	→	0	3434	6432	0	1
0	c_2	p_2	→	0	280	17424	0	1
1	c_2	p_2	→	12	280	17424	0	1
2	c_2	p_2	→	24	280	17424	0	1

Encryption The encryption operator hides target fields by applying a symmetric encryption function to one or more fields. The encryption operator is denoted $E_{X(Y),\kappa}$ where X is a set of target fields to be encrypted, Y is an optional set of grouping attributes for encryption, and κ is a secret encryption key.

The encryption operation is applied as follows. For each record in the trace, the values of attributes from set X are concatenated with the values of attributes from set Y to form a string. The string is appropriately padded and then encrypted under a symmetric encryption algorithm using κ as the key. The ciphertext output replaces the fields of X in the output trace; the values for attributes in Y are not affected.

The encryption key is never shared, so the output trace must be analyzed without access to the values in these fields. A different encryption key is used for each encryption operator applied, but the same encryption key is used for all values of the fields in X. Thus, common values in an encrypted field are revealed to the analyst. However, if two records agree upon values in X but differ in values in Y, then the encrypted values of X will be different for these records. As a result, the encryption of two records will be same only if they agree upon values for X as well as for Y.

Table 2(b) shows the result of applying encryption operators $E_{\{ip1,ip2\},\kappa}$ and $E_{\{pt1,pt2\}(ip1,ip2),\kappa}$ to Table 2(a). The encryption allows connections (identified by source and destination IP, port fields) to be differentiated. However, it is not possible see that two connections share the same destination port, for example. Further, because source and destination IP are used as input for encryption of ports, it is not possible to correlate ports across different connections.

Canonical Ordering The *canonical ordering* operator is used to replace fields whose actual values can be eliminated as long as they are replaced by synthetic values respecting the ordering of the original values. The ordering operator is denoted $O_{X(Y)}$ where X is the set of target fields to be replaced, and Y is an optional set of grouping fields. If the input set Y is empty, the data entries in fields of X are sorted and replaced by their order in the sorted list, beginning with one. If the input set Y is not empty, then the ordering operation is done separately for each group of records that agree on values for the columns in Y.

Translation The translation operation is applied to numerical fields in the trace, shifting values through addition or subtraction of a given constant. The operator is denoted $T_{X(Y)}$ where X is a set of target columns that are translated by the operator. The operator can optionally have another set of columns Y called grouping columns, which are not affected by the operation.

If the input set Y is empty, all the data-entries in target columns in X are shifted by a parameter c. The shift is caused by subtracting a random parameter c from each entry in the columns. If the input set Y is not empty, then all the records in a trace are formed into groups such that the records in each group have the same data for columns in Y. For records in each group, the target columns X are shifted by a parameter c where the value of the parameter is dependent on the group. The parameter value can be chosen randomly or by using a function that takes the data-entry of Y for the group as input.

Scaling The scaling operation scales all the values in a given field by multiplying it with a constant multiplier. The scaling operator is denoted $S_{X,k}$ for a set of target fields X. The scaling operator acts scales all the values in fields in X by a factor of k.

It is sometimes convenient to consider the identity transformation, denoted I_X, which does not transform field X, including it in the output without modification.

Composite Transformations

The operators above can be combined to form composite transformations for a trace. We assume in the sequel that composite transformations ϕ are represented in the following normal form:

$$\phi = \Pi_X \circ \ \phi^1_{X_1} \circ \phi^2_{X_2} \circ ... \circ \phi^n_{X_n} \tag{1}$$

where $\phi^i_{X_i}$ refers to $(i+1)^{th}$ operator in ϕ which acts on attribute set X_i and for all i, $\phi^i_{X_i} \in \{E, T, O, S, I\}$. We denote the set of all such transformations Φ. The last operation applied to the trace is the projection Π_X. Any operator acting on fields not present in X will be disregarded. Further we restrict our attention to composite operations in which each field in the trace is affected only by one operation: $\forall i, j, X_i \cap X_j = \{\}$. In the paper, we will assume Π_X to be present even if not mentioned in ϕ. For example, $E_{X_1} \circ T_{X_2}$ and $\Pi_{X_1 \cup X_2} \circ E_{X_1} \circ T_{X_2}$ will be the same.

Other operators.

Our framework can easily accommodate other transformation operators. We have found that this simple set of operators can be used to generate safe transformations supporting a wide range of analyses . In many cases, adding additional transformation operators to our framework requires only minor extensions to the algorithms described in Section 4. For example, network researchers have proposed a prefix-preserving encryption of IP addresses which could be added as a special transformation operator in our framework. However, it is worth noting that some potentially important operators (e.g. random perturbation of numeric fields, or generalization of field values) will lead to analysis results that are approximately correct, but not exact. In this initial investigation, we are concerned with supporting exact analyses. We leave as future work the consideration of such operators and the evaluation of approximately correct analysis results.

3.2 Specifying Utility Requirements

In our framework, the analyst seeking access to a trace must specify their utility requirements formally. These requirements are expressed as a set of *constraints* asserting a given relationship between fields in the original trace and fields in the anonymized trace. The analyst is expected to specify constraints that are sufficient to allow the exact analysis to be carried out on the trace. Each con-

Table 3: *Lookup table for unary constraints used for verifying utility of transformation*

expression	**transformations**
t.a ≤ t.b t.a ≥ t.b	$T_{X(Y)}, I_X, S_{X,k}$ $\{a,b\} \subseteq X$
$t.a == t.b$ $t.a! = t.b$	$T_{X(Y)}, S_{X,k}, E_{\{a\}(Y),\kappa} \circ E_{\{b\}(Y),\kappa}, I_X$ $a \notin Y$,$b \notin Y$, $\{a,b\} \subseteq X$,
$t.a - t.b$	$T_{X(Y)}, I_X, \{a,b\} \subseteq X$
$t.a + t.b$	$I_X, \{a,b\} \subseteq X$
$t.a \times t.b$	$I_X, \{a,b\} \subseteq X$
$t.a/t.b$	$S_{X,k}, I_X, \{a,b\} \subseteq X$
t.a	$I_X, a \in X$

straint states which item of information must be preserved while anonymizing the trace. An item of information in a trace can be either: (i) the value of some field in the trace, or (ii) the value of some arithmetic or boolean expression evaluated using the fields from the trace only. The syntax of notation for the constraint is as follows:

DEFINITION 1 (UTILITY CONSTRAINT). *A utility constraint is described by a rule of the following form:*

$$\langle \textit{qualifier} \rangle \Rightarrow (expr(orig) = expr(anon))$$

where $'expr'$ can be any acceptable arithmetic expression obtained using operators $(+, -, /, *)$ or it can be any boolean expression obtained using boolean operators $(\&\&, ||, \leq, \geq, ==, ! =)$ on fields available in the trace. The $expr$ does not involve any numeric constants. We use record.field and ϕ(record).field, to mean any valid field in the original network trace and the transformed trace, respectively. The 'qualifier' in the constraint is set of boolean conditions that must be true for qualifying records.

The above constraint means that if there are one or more records in a trace that satisfy the qualifying condition given in $\langle \textit{qualifier} \rangle$, then the value of expression $expr$ evaluated over these records must be equal to the value of same expression when evaluated over corresponding anonymized records.

A constraint rule is *unary* if its conditions refer to a single record. For example, if an analyst wants to test two port numbers for equality, this can be expressed as the following unary constraint:

$$Any(t) \Rightarrow ((t.pt1 == t.pt2) = (\phi(t).pt1 == \phi(t).pt2))$$

The qualifier $Any(t)$ is true for any record in the the trace. The *constraint* says that if the two port numbers in a record have same value then the corresponding values in the anonymized record should also be the same. We can see that the information that needs to be preserved is the equality of port numbers and not their actual values.

A *binary constraint* requires two records for evaluating its expression. For example, the analyst may want to verify that the acknowledgement number in one packet is equal to the sequence number of another packet involved in the same TCP connection. This requirement can be expressed as following constraint:

$$Same(t1, t2) \Rightarrow (t1.ack == t2.seq) = (\phi(t1).ack == \phi(t2).seq)$$

The information that needs to be preserved here is the equality of two fields across records. The actual values need not be preserved. The qualifier $Same(t1, t2)$ is user-defined and it states the conditions that must be true for two records to belong to the same connection. In this case, the list of conditions is $\{(t1.ip1 == t2.ip1)$, $(t1.ip2 == t2.ip2)$, $(t1.pt1 == t2.pt1)$, $(t1.pt2 == t2.pt2)\}$.

We believe trace analysts will be able to use these constraint rules to accurately describe the properties of a trace required for accurate analysis. In most cases it is not difficult to consider a trace analysis and derive the fields whose values must be unchanged, or the relationships between values that must be maintained. (See [13] for additional examples.) The analyst could be assisted in this task by a GUI or semi-automated procedures, but this is beyond the scope of the current work.

We note that it is in the interest of the analyst to choose a set of constraint rules which are specific to the desired analysis task. Obviously, if all fields in the trace are required to be unmodified, then the only satisfying trace will be the original, unanonymized trace. Our framework does not impose any explicit controls on the utility requirements submitted by analysts, except that the trace owner is likely to reject requests for constraint requirements that are too general.

4. ANALYSIS OF TRANSFORMATIONS

An important feature of our framework is that it enables the trace owner to reason formally about the relationship between utility requirements and anonymizing transformations. In this section we show how the trace owner can determine conclusively that a published trace will satisfy the utility requirements expressed by analysts. In addition, we show how the trace owner can derive the *most secure transformation* satisfying a desired set of utility constraints. Lastly, we show how the trace owner can compare alternative publication strategies and analyze the potential impact of collusion amongst analysts who have received traces.

We refer to the formal reasoning about trace transformations and utility constraints as *static* analysis because these relationships between transformations hold for all possible input traces. Other aspects of trace privacy cannot be assessed statically; in Section 5 we measure the privacy of real traces under sample transformations.

4.1 Verifying the utility of a transformation

We now show that it is possible to test efficiently whether a given transformation will always satisfy the utility requirements expressed by a set of constraints.

DEFINITION 2 (UTILITY CONSTRAINT SATISFACTION). *Given a set of constraints C and a transformation ϕ, we say ϕ satisfies C (denoted $\phi \models C$) if, for any input trace, the output of the transformation satisfies each constraint in C.*

Checking utility constraint satisfaction is performed independently for each constraint rule in C by matching the conditions specified in a constraint to the operators that impact the named fields. Recall that the general form for unary constraints is $\langle \textit{qualifier} \rangle \Rightarrow (expr(t) = expr(\phi(t)))$ where $expr$ can either be conjunctive normal form of one or more comparisons, or an arithmetic expression. Since the unary constraint has only one record, each comparison expression must involve two attributes from the network trace. For each comparison or arithmetic expression in $expr$, we look for the corresponding entry in Table 3 which lists expressions and compatible transformation operators. If the composite transform function ϕ has a matching transformations in Table 3, then we proceed to the next comparison or sub-expression. Otherwise we conclude that ϕ does not satisfy the constraint. If ϕ has a matching transformation for each of the sub-expressions, the constraint is said to be satisfied by the transformation.

The procedure for verifying binary constraints is similar, with some minor modifications. The details of the verification process can be found in our technical report [13].

4.2 A partial order for transformations

Since each transformation operator removes information from the trace, some composite transformations can be compared with one another in terms of the amount of information they preserve. We show here that there is a natural partial order on transformations.

DEFINITION 3 (INVERSE SET OF A TRANSFORMED TRACE). *Let $\mathcal{N}$ be a trace transformed using transformation ϕ to get trace $\phi(\mathcal{N})$. Then, the inverse set for trace $\phi(\mathcal{N})$ is given by all possible traces that have the same algebraic properties as in $\phi(\mathcal{N})$ and hence, can give $\phi(\mathcal{N})$ as output when transformed using ϕ. We use notation $\phi^{-1}[\mathcal{N}]$ to represent this set.*

DEFINITION 4 (STRICTNESS RELATION, $\preceq$). *Given two composite transformations ϕ_1 and ϕ_2, we say that ϕ_1 is stricter than ϕ_2 if -*

$$\forall \textit{ Trace } \mathcal{N},\ \phi_2^{-1}[\mathcal{N}] \subseteq \phi_1^{-1}[\mathcal{N}]$$

The definition says that if the transformation ϕ_1 is stricter than ϕ_2, then the inverse set for any trace $\mathcal{N}$ transformed by ϕ_1 is bigger than that of ϕ_2. In other words, ϕ_1 leads to more traces to look similar to original trace $\mathcal{N}$ and hence, it contains less information about the original trace as compared with ϕ_2

Using the definition of strictness, the most strict transformation is $\Pi_\emptyset$, which removes all attributes of the trace. The least strict transformation is $I_{\bar{X}}$, which simply applies the identity transformation to all attributes, returning the original trace without modification. All other transformations fall between these two in terms of strictness. For example, we have $E_{X(Y)} \prec O_{X(Y)}$ because encryption removes the ordering information from the data-entries. Also, $E_{X(Y')} \prec E_{X(Y)}$ if $Y \subseteq Y'$ as $E_{X(Y')}$ removes the equality information of X-entries from records which have the same entries for Y but differ in some attribute in $(Y' - Y)$. More strictness relations for basic operators are given in our technical report [13].

Recall that Φ denotes the set of all composite transformations. Then the following theorem show that the strictness relation has a number of convenient properties.

THEOREM 1. *$(\Phi, \preceq)$ is a partially ordered set and forms a join-semilattice i.e. for any two transformations ϕ_1 and ϕ_2, there is another transformation in Φ, denoted lub(ϕ_1, ϕ_2), which is the least upper bound of ϕ_1 and ϕ_2.*

The proof of this result is included in our technical report [13] .

Theorem 1 can be easily extended to conclude that any set of transforms has a unique least upper bound. This fact has a number of important consequences for the trace publisher:

Maximizing trace security Given a set of constraints C it is important for the trace publisher to compute the *most secure transformation* satisfying C. Theorem 1 shows that such a transformation always exists. We describe an algorithm for computing this transformation in the next subsection.

Combining transformations Imagine that the trace publisher has derived three transforms ϕ_1, ϕ_2, ϕ_3 specific to three analyst requests. The publisher may wish to consider publishing a single trace that can satisfy all three requests simultaneously. The least upper bound of these three transformations, denoted $lub(\phi_1, \phi_2, \phi_3)$ is the transformation with least information sufficient for all three analysts.

Collusion analysis Similarly, if the publisher has already released the traces derived from ϕ_1, ϕ_2, ϕ_3 and fears that the analysts may collude, then the least upper bound transformation $lub(\phi_1, \phi_2, \phi_3)$ is a conservative bound on the amount of information the colluding parties could recover by working together. We discuss collusion analysis in Sec 4.4.

4.3 Most Secure Transformation

The existence of least upper bound of any two transformation allows us to find the most secure transformation that can satisfy a given set of utility requirements.

DEFINITION 5 (**MOST SECURE TRANSFORMATION**). *Given a set of constraints C, the most secure transformation is the minimum element in* $\Phi[C]$*, denoted* $\min(\Phi[C])$.

We denote by $\Phi[C]$ the set of transformations satisfying the constraints of C. Algorithm 1 computes the most secure transform given a set of constraints. The algorithm uses a map data-structure which keeps the mapping of a set of constraints to its most secure transform. It starts by forming $|C|$ different constraint sets with each set having exactly one constraint. Using the look-up table for constraints, the strictest operator is obtained for each constraint and the entry is made in the map (Lines 3-8).

As a next step, two constraint sets (C_1, C_2) are chosen such that there exist an attribute which is referred by at least one constraint in each set. The composite transforms for C_1 and C_2 can operate differently on this common attribute. Thus, the least upper bound of these transforms is computed to get the most secure transform having properties of both the transforms (Lines 11-13). The steps for obtaining lub can be seen in the proof of Theorem 1 (refer [13]). The constraint sets C_1 and C_2 are now merged to obtain a single set and it is put into the map along with lub. The previous entries for the two sets are removed from the map. (Lines 14-16).

The above steps are repeated until no dependent constraint sets are left. Now, all the transforms in the map transform disjoint set of attributes and do not conflict. As a final step, the composition of all these transforms is done. The resulting composition operator along with the required projection operator is returned as the most secure transform (Lines 18-23).

Algorithm 1 Most Secure Transform

```
Input: Set of Constraints $C$
Output: Composite Transform
Let S = { } be empty set of attributes
Let $map$ = { } be Constraint-set to Transformation Map
for all constraint $c$ in $C$ do
    for all attribute $a$ present in $c$ do
        $S = S \cup \{a\}$
    end for
    $\rho$=Most Secure Operator from look-up table that satisfies $c$
    PUT(map,{c},$\rho$)
end for
while $\exists$ dependent sets $C_1, C_2$ in $map$ do
    $\rho_1$ = GET($map$,$C_1$)
    $\rho_2$ = GET($map$,$C_2$)
    $\rho$ = LEAST-UPPER-BOUND($\rho_1, \rho_2$)
    REMOVE($map, C_1$)
    REMOVE($map, C_2$)
    PUT($map, C_1 \cup C_2, \rho$)
end while
$\phi$ = $\prod_S$
for all set $C$ in $map$ do
    $\rho$= GET($map$,$C$)
    $\phi = \phi \circ \rho$
end for
return $\phi$
```

4.4 Evaluating collusion risk

Customizing published traces to the needs of analysts means that any given analyst will have the information they need, but not more than they need. However, if a trace owner publishes a set of traces, we must consider the implications of one party surreptitiously acquiring multiple traces and attempting to combine them.

The ability to correlate the information present in two (or more) transformed traces depends greatly on the particular transformations. As a straightforward response to the risks of collusion, the trace owner can always consider the least upper bound, *lub*, of the published transformations. The *lub* provides a conservative bound on the amount of information released since each of the published traces could be computed from it. Therefore the trace owner can evaluate the overall privacy of publishing the *lub* transformation; if it is acceptable, then the risk of collusion can be ignored.

In many practical cases the *lub* is overly conservative because judging the risk of disclosure based on lub assumes that the released traces can be accurately combined. In many cases, this can't be done. For example, if the two transformations $\sigma_1 = E_{\{ip1,ip2\},\kappa_1} \circ I_{TTL}$ and $\sigma_2 = E_{\{ip1,ip2,pt1,pt2\},\kappa_2} \circ I_{win}$ are released, it is difficult to correlate them to obtain the least upper bound given by $lub(\sigma_1, \sigma_2) = E_{\{ip1,ip2\},\kappa_3} \circ E_{\{pt1,pt2\},\kappa_4} \circ I_{win} \circ I_{TTL}$. One trace includes a TTL field for each pair of hosts, while the other includes window field for each encrypted connection. Because distinct port pairs are included in the encryption in the second trace, but removed from the first, it would be very difficult to relate records across the two traces. In general, the relationship between the information content present in two transformations σ_1, σ_2 and the information present in the $lub(\sigma_1, \sigma_2)$ depends on (i) the transformation operators applied to fields common to σ_1 and σ_2, and (ii) the degree to which these fields uniquely identify packets in the trace. Static analysis tools can be used to evaluate (i), however (ii) may depend on the actual input trace and requires quantitative evaluation, similar to that described next.

A more accurate assessment of collusion requires the quantitative analysis of the traces released. Intuitively, the released traces are similar to decompositions of the original trace table (but with complex transformations applied to its fields). In many cases, the decompositions are lossy, making it difficult to combine the released traces. The merge process gets even more difficult as the common fields available in these traces may be transformed in different ways removing different pieces of informations from them. Thus, the merging of traces not only requires common fields but also, understanding the common information available in them. If two traces share common fields, then the common information available in both is given by the greatest lower bound *glb* of the corresponding transformations. As an example, *glb* of two transformations $\sigma_1 = E_A \circ T_B$ and $\sigma_2 = O_A \circ T_C$ is given by $glb(\sigma_1, \sigma_2) = E_A$. Thus, these two traces can be merged using only information retained by encryption operator E_A. The order information of A available in σ_2 cannot be used for relating records as the information is not present in σ_1.

We now show how the *glb* can be used to merge any two transformed views of the trace. The following steps allows us to quantify the success with which records in two views can be merged. Also, it provides us with probabilistic bounds on the possibility of relating records across views accurately.

Let us suppose, we have two traces obtained by applying transformations σ_1 and σ_2 to a trace. As a first step towards merging, we project only the common fields from σ_1 and then, compute common usable information for merge from it using $glb(\sigma_1, \sigma_2)$ to obtain a view V_1. Similarly, we obtain common usable information from σ_2 as view V_2. The two traces σ_1 and σ_2 can be accurately combined only if we have one-to-one mapping from records in V_1 to records in V_2.

If none of the fields in glb is encrypted, the process of mapping records from V_1 to V_2 is straightforward. First, we look for the records in V_1 which have unique values. As these values are not encrypted, we can find the corresponding records with same values in V_2. Thus, we can map these records accurately. The percentage of such records in the view gives the success rate of collusion. For records with non-unique values, we partition them into groups of records with same values together. For a record in each group, it can be mapped to its corresponding record in another view correctly only with probability given by $1/n$ where n is the size of the group. The chance of successful collusion is limited if the percentage of exact matches is small and the probability of matching unmatched record is bounded to limit $1/l$ where $l > 1$.

The process of mapping records is more complex if some of the fields in *glb* are encrypted. This is because the encrypted values cannot be compared in two views obtained using different encryption keys. However, the encryption operator preserves the count of records that share the same values on encrypted attributes. Recall that the encryption operator is of form $E_{X(Y)}$ where Y is set of grouping attributes and may not be encrypted. As a first step in mapping process, we partition the records in V_1 by grouping the records which have same values for all Xs and Ys used in encryption operators in *glb*. We do similar partitioning in V_2. Next, for each group in V_1, we note the count of records (n) in that group. Now, for each operator $E_{X(Y)}$ in *glb*, we look at each group, observe the values of X and Y in it, and count the records in V_1 that have the same values as that group. We maintain these counts along with n for each group. We repeat these steps for the view V_2.

The next step is to map each group in V_1 with its corresponding group in V_2. As the encrypted values in these views cannot be compared, we use the counts computed previously to compare groups in V_1 and V_2. It is possible that a group in V_1 may have more than one candidate groups in V_2. Next, we use the values of un-encrypted fields to compare a group in V_1 with its candidate groups in V_2. We discard those candidates which do not agree on these fields. If we are left with only one candidate group, we map the records in group of V_1 to the records in its candidate group. The mapping process is similar to the case when encryption is not used. However, only unencrypted values must be used to find the mapping. These steps should be done for each group in V_1. The percentage of records mapped uniquely gives the success of collusion. A record is unmatched if there is more than one candidate for its group or it has non-unique values in its group. The probability of mapping it accurately in first case is bounded by $1/m$ where m is the number of candidates for its group. In the second case, the probability can be computed as described earlier.

After performing the analysis of collusion, we can quantitatively assess the certainty with which two transformed traces can be matched. We may conclude, for example, that because large number of records in V_1 have at least k possible matching records in V_2, that the collusion risk is negligible. In the next section we describe the results of this quantitative evaluation for real communication traces.

5. QUANTIFYING TRACE PRIVACY

The techniques in the previous section allow the trace owner to compare the information content of some transformations, and to find the most secure transformation that satisfies a given set of utility requirements. This provides a *relative* evaluation of trace privacy. In this section, we perform a quantitative analysis of the traces, which serves a few purposes. First, it allows us to compare the transformations that are not comparable (recall that these cases occur because the relation $\preceq$ in Section 4 is only a partial order). Second, we can validate the strictness relations among the transformations. Third, it provides the trace owner with a measure to evaluate the privacy of the view. In addition, the quantitative analysis can be used to assess the implications of possible collusion attacks.

Evaluation Model

The most common threat in trace publication is the re-identification of the anonymized entities. This is done using a fingerprinting attack. In this attack, the adversary tries to re-identify the entities using information (available through external means) about properties of the entities participating in the collected trace. These properties are called fingerprints. To evaluate the trace security, we collect all the available fingerprints from the original trace and try to match them with the entities in the anonymized views. The fingerprint matching is done using a similarity metric. The anonymized entities that have similarity metric above a certain threshold are added to the candidate set of the entity being matched. Then, we measure privacy of the trace by computing, for various k, the value of $\mathcal{N}(k)$: the number of entities in the trace that have a candidate set of size less than or equal to k. For example, out of the total number of entities in the trace, $\mathcal{N}(1)$ indicates the number of entities that are uniquely identifiable. Clearly, a lower value of $\mathcal{N}(k)$ indicates a more privacy-preserving trace.

Experimental setup

In our privacy evaluation, we have used a single IP packet trace collected at a gateway router of a public university. The trace consists of 1.83 million packets and has 41930 hosts, both external and internal. The trace was stored as a relational table in the IBM DB2 Express-C 9.5 database system running on a Dell workstation with Intel Core2 Duo 2.39 GHz processor and 2GB RAM.

The evaluation was done over five anonymized views obtained using transformations given in Figure 2(a), which were motivated by some of the analyses studied in the literature. The first two transformations σ_1, σ_2 support the example analysis given in Section 2. The transformations ψ_1 and ψ_2 allow the analyst to count the number of entities which are involved in connections with a high rate of out-of-sequence packets. The transformation ϕ_0 has been chosen as a base case such that it is an upper bound of both σ_1 and ψ_1. As such, it is capable of supporting both the above-mentioned analyses. Figure 2(b) illustrates the strictness relationships between the five transformations studied here.

As fingerprints of each entity in trace, we have extracted information about its open ports, the entropy information of the entities it communicates with, entropy of ports used in communication, duration of communication and additional information on fields from trace like maximum and minimum value of TTL and signature chain of sequence numbers. We have used standard similarity metrics like Jaccard coefficient, Pearson correlation coefficient to match the fingerprints. We chose a high threshold value of 0.98 to enforce strong privacy requirements.

Results and Conclusion

We have summarized our results in Figure 3. Based on the privacy measure $\mathcal{N}(1)$, the number of uniquely identified entities, the following conclusions can be made from the experiment-

- The least strict transformation ϕ_0 is the least secure transformation which led to re-identification of 1904 of the 41930 entities present in the trace.

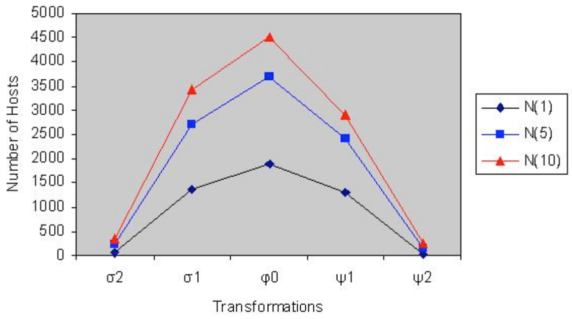


Figure 3: *Results giving number of hosts, $\mathcal{N}(k)$, whose candidate set size was $\leq k$ for $k = 1, 5, 10$*

- The strictness relations among the transformations as shown in Figure 2(b) are validated.
- The significance of fields for information disclosure about entities can be observed. For example, the significant difference in $\mathcal{N}(1)$ value for ψ_1 and ψ_2 indicates that the entropy of individual ports is highly informative.
- The low value of $\mathcal{N}(1)$ for σ_2 and ψ_2 (66 and 25 respectively) as compared to general view ϕ_0 (1904) illustrates the significant gain in anonymity resulting from publishing two traces customized for individual analyses. It allows the trace owner to decide against publishing a single general view.
- The statically incomparable transformation ψ_2 and σ_2 can be compared using this measure. The lower $\mathcal{N}(1)$ value for ψ_2 as compared to σ_2 indicates that ψ_2 is more secure than σ_2.
- For an analyst with view σ_2, if he colludes with analyst in possession of ψ_2 then the only additional information he learns is order information of $IPID$ field. This field does not lead to high information disclosure as evident from low $\mathcal{N}(1)$ value for ψ_2. Thus, the collusion does not lead to significant risk.

In a different quantitative analysis, we measure the efficacy of collusion of views ψ_2 and σ_2. As described in Section 4.4, we group the records in one of the view on encrypted fields common to both. Then, we identify the groups that have a single matching group in other view due to their unique size or have unique set of values for other common fields. The records in these groups can be joined using the order information available in common fields. The evaluation showed that significant number of records (30.6%) could be joined with full certainty without even using the information from syn and ack fields. But as shown earlier, the information disclosure after successful collusion is not high.

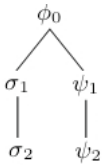
Discussion

Using a quantitative analysis, like that described above, the trace publisher may decide that some transformed traces are not safe for release. In this case, our framework would recommend that the request for such traces be denied because the utility constraints cannot be satisfied under acceptable privacy conditions. This reflects our objective to release traces that provide perfect utility to analysts.

If, however, the analyst is willing to sacrifice accuracy, the trace publisher can consider alternative anonymization techniques as a subsequent step. Such alternative anonymization techniques are not the focus of the present work, although we believe applying our transformation prior to further anonymization is valuable: our

$\sigma_1 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ E_{\{pt1\},\kappa_2} \circ E_{\{pt2\},\kappa_2} \circ T_{\{ts\}(C)} \circ T_{\{seq,ack\}(C)} \circ I_{\{win,dir,syn,ack\}}$
$\sigma_2 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ E_{\{pt1,pt2\},\kappa_2} \circ T_{\{ts\}(C)} \circ T_{\{seq,ack\}(C)} \circ I_{\{win,dir,syn,ack\}}$
$\psi_1 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ E_{\{pt1\},\kappa_2} \circ E_{\{pt2\},\kappa_2} \circ O_{\{ts\}(C)} \circ O_{\{seq\}(C)} \circ O_{\{ack\}(C)} \circ O_{\{ipid\}(C)} \circ I_{\{dir,syn,ack\}}$
$\psi_2 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ E_{\{pt1,pt2\},\kappa_2} \circ O_{\{ts\}(C)} \circ O_{\{seq\}(C)} \circ O_{\{ack\}(C)} \circ O_{\{ipid\}(C)} \circ I_{\{dir,syn,ack\}}$
$\phi_0 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ T_{\{ts\}(C)} \circ T_{\{seq,ack\}(C)} \circ I_{\{pt1,pt2,win,TTL\}} \circ O_{\{ipid\}(C)} \circ I_{\{dir,syn,ack\}}$

(a)



(b)

Figure 2: *(a) Five example transformations used in the quantitative evaluation of individual anonymity. (b) Tree representing strictness relationships between the example transformations (i.e. [child] $\preceq$ [parent] in each case).*

transformations remove or obscure the information and relationships among records that are not needed for the intended analyses, allowing techniques like cell suppression, generalization, or perturbation to be applied only where truly necessary.

It may also be possible to use our quantitative analysis as input to further anonymization. For example, from the quantitative analysis, the trace publisher knows exactly the entities in the trace that can be identified uniquely. One simple approach to make the trace secure is to delete all the records belonging to these entities before publishing. The impact on the analysis accuracy upon record deletion depends entirely on the nature of analysis. In an alternative approach, the trace publisher runs the quantitative analysis with a lower threshold on similarity of fingerprints. The lower threshold represents a weaker adversary in quantitative analysis and may result in vulnerable entities to have more than one entity in its candidate set. Now, for each such entity, we modify its fingerprints available in the trace by removing one or more connections of this entity. The connections are removed in such a way so that similarity of the actual fingerprints and the fingerprints available in the trace after deletion becomes less than or equal to the new threshold chosen for analysis. For the entities that are still uniquely identifiable with the lower threshold, the process can be repeated using newer threshold value or its records can be deleted from the trace. This approach will result in fewer record deletions, and higher utility than the previous approach.

6. SYSTEM IMPLEMENTATION

In this section we propose techniques for the trace owner to efficiently transform large traces in response to multiple requests from analysts. We use a relational database to store the original trace and to apply transformations, creating new traces to be released to users.

If a composite transformation only uses operators from (E, Π, T, S, I), then the transformation can be executed as a single scan of the trace. Relative to the scan, operators I and Π are costless. The operators E, T, S, can be implemented efficiently using scalar user-defined functions (UDFs), and add only modest CPU overhead.

On the other hand, the implementation of the canonical ordering operator $O_{X(Y)}$ is not trivial. Recall that this operator groups data records on attributes in Y, and then replaces the values in X by their corresponding ranks in its group. The grouping and rank computation requires sorting of data in order $\{Y, X\}$, leading to multiple passes over the data, thus potentially incurring significant IO cost.

After describing a baseline implementation, we focus on optimizing the implementation of the operator $O_{X(Y)}$. We describe an approach based on storing redundant projections of the trace to speed up sorting, and an approach based on memoization, which exploits repeated or related requests for transformations.

6.1 Baseline Implementation

Our operator $O_{X(Y)}$ is similar to the DENSE_RANK() function recently added to the SQL:2003 standard. This function computes the rank of the tuples in a relation based on the rank criteria provided in assisting clause. The operation $O_{X(Y)}$ can be done using following clause: DENSE_RANK OVER (PARTITION BY Y ORDER BY X).

In the literature, a lot of work has been done in the area of efficient rank computation. However, it is not relevant to our problem because all these works assume that there can be only one associated rank with a tuple. Most of these are concerned with returning top-k tuples or performing efficient join of two relations with different rankings to obtain a single relation with combined ranks. In our case, we have only single relation and we can have multiple ranks for a tuple due to multiple O operators in the transformation. Also, we return all the tuples and not just top-k.

As an example, let us consider the following composite transformation:

$$\phi = E_{ip1,ip2} \circ T_{seq\_no,ack\_no(ip1,ip2)} \circ O_{ts(ip1,ip2)} \circ I_{window}$$

Assuming that the trace has been stored in the relation *TRACE* and we have the required UDFs, the equivalent SQL query is:

```
SELECT ENCRYPT(ip1,ip2), TRANSLATE(seq_no,ip1,ip2),
TRANSLATE(ack_no,ip1,ip2), window, DENSE_RANK() OVER
(PARTITION BY ip1,ip2 ORDER BY ts) as ts
FROM TRACE;
```

Any general transformation ϕ can be seen as composition $\phi^{[S]} \circ \phi^{[O]}$ where $\phi^{[S]}$ consists only of scalar operators E, T, S, I and $\phi^{[O]}$ consists only of O operators. As the cost is dominated by $\phi^{[O]}$, we will focus on the cost analysis for $\phi^{[O]}$. Let us consider general composite transformation $\phi^{[O]}$ consisting of n O-operators. Let this be:

$$\phi^{[O]} = O_{X_1(Y_1)} \circ O_{X_2(Y_2)} \circ ... \circ O_{X_n(Y_n)}$$

The SQL query for this transformation can be easily written using n *DENSE_RANK()* clauses. The query plan for this query requires the sequential scan of relation *TRACE* to begin sorting in order $\{Y_1, X_1\}$ for the first rank clause. The sorted relation is written into temporary table consisting only of desired columns. This is followed by $n - 1$ consecutive sorts of temporary table where each sort order is determined by each rank clause in the query. The temporary table with ranks is returned as the query result.

If the relation *TRACE* consists of N pages, the temporary table may have fewer columns and may occupy a fraction f $(0 < f \leq 1)$ of N pages. We will use the term *projection-ratio* to mean the fraction f. If B be the number of pages available in the buffer pool, then the cost of the query plan is given by:

$$C = N + n \times S(fN, B) \qquad (2)$$

where the cost of sorting is $S(N, B) = 2N(1+\lceil \log_{B-1} \lceil N/B \rceil \rceil)$.

It can be seen that the cost increases linearly with number of rank operators in the query. Also, the cost increases linearly with

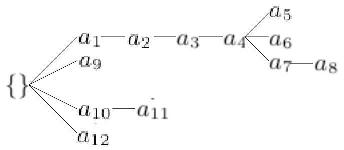


Figure 4: Workload Tree

size of data N in the range $B < N < B(B-1)$. Figure 6(a) shows the experimental verification of linear increase by varying the number of rank operators and the size of data from 20 million to 52 million records. The data consisted of IP packet trace collected at the gateway router of a public university. We have used freely available version of IBM DB2 Express-C for the experiments.

6.2 Cost Optimization

Next we describe a cost reduction technique using vertical partitions. The vertical partition of a relation R is a projection of R which contains an additional column with tuple identifier tid. The tid allows the tuple in partition to be associated with the parent tuple in R. For any general transformation, different vertical partitions can be selected such that each partition computes some of the ranks required in the transformation. The rank computation in each partition is done using the approach stated in previous subsection. The tuples containing ranks across partitions can then be equi-joined using tid to obtain the transformed relation. This approach results in cost reduction as it can avoid redundant sorting of columns that are not required in the rank computation. However, it incurs additional cost of sorting on tid to do sort-merge join of partitions. We explain the cost reduction using an example.

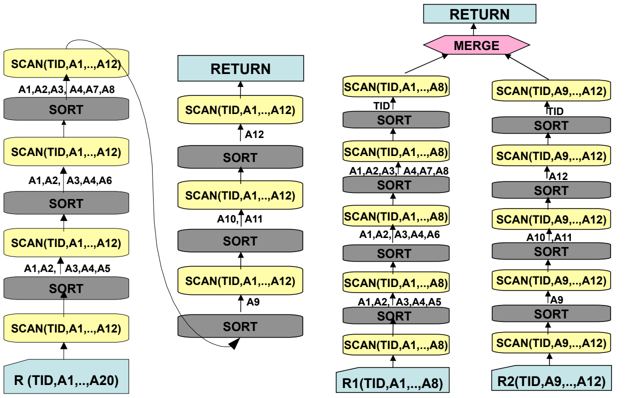
DEFINITION 6 (WORKLOAD TREE). *For a given transformation, the workload tree is a prefix-tree that represents the sort-orders required for the O operations. Each leaf in the tree corresponds to an O operator and the path from root to leaf gives the corresponding sort-order.*

In Figure 4, we present a workload tree for a transformation with six O operators. Figure 5(a) shows a naive query plan over a single relation R for this workload. Note that columns $a_9, a_{10}, a_{11}, a_{12}$ have to undergo 6 sorting phases even if only 3 sorts are required to compute 3 ranks on these columns. On the other hand, Figure 5(b) shows a query plan using two partitions. In this query plan, the data in columns $(a_1, a_2, ..., a_8)$ and $(a_9, ..., a_{12})$ have to go through only 4 sorting phases as opposed to 6 in naive approach. Thus, vertical partitions save us from redundant sorts and reduce costs by sorting on smaller partitions.

The cost computation of this approach is similar to naive approach. It is the sum of naive approach costs for each partition used and the cost of doing sort-merge join of all the partitions. If P is the set of partitions chosen for query evaluation, let n_i be the number of rank operators supported by partition $p_i \in P$, $f_i N$ be the pages occupied by p_i where $0 < f_i \leq 1$ and N is number of pages for relation R. Additionally, there is projection ratio g_i associated with p_i. Using the eqn (2) for each partition cost and $\Sigma_P S(f_i g_i N, B) + f_i g_i N$ as cost of sort-merge join, overall cost is given by

$$\Sigma_P (f_i N + n_i \times S(f_i g_i N, B) + (S(f_i g_i N, B) + f_i g_i N)\{|P| > 1\}) \quad (3)$$

From our experiments, we have observed that the savings estimated using above equation is within error bounds of 5.12%. Figure 6(b) shows the reduction in transformation time for workload



(a) (b)

Figure 5: (a)Naive query plan for a query with six rank operators (b) Query plan using two vertical partitions of the relation

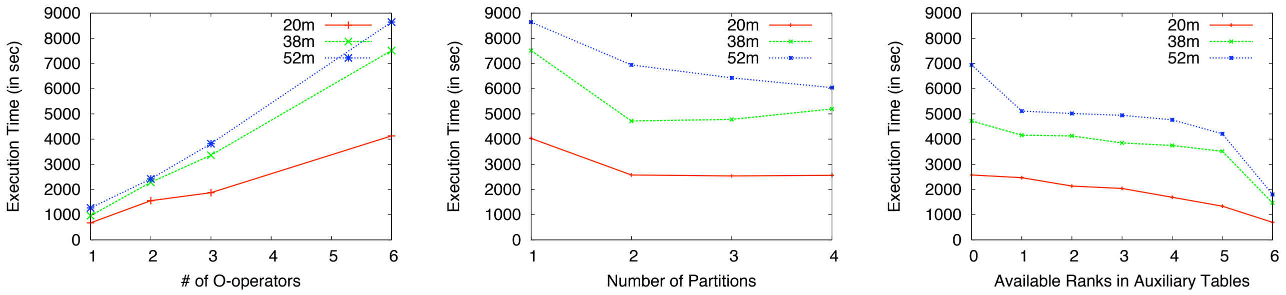
described in Figure 4 as the number of partitions in query plan increases. However, increasing the number of partitions beyond 4 led to an increase in transformation time as the cost of merging partitions begins to dominate. Thus, it is important to choose partitions and distribute rank-computations across them optimally. It is possible to use standard search algorithms like A*-search to determine this optimal partitioning for a given workload in a cost-based manner using eqn(3).

6.3 Memoization

The communication traces are immutable in nature i.e. there are no updates once the trace has been collected. Hence, any canonical ordering computed can be stored and used later in order to save us from some work in applying newer transformations. In general, the total number of possible canonical orderings is exponential in the number of columns in traces. However, we have observed that the set of interesting canonical orderings for the analysts is much smaller. Thus, it is likely that different transformations can have common canonical orderings in them. These transformations are combination of orderings from the interesting set.

In order to save us from the repeatedly computing the same orderings, we maintain an auxiliary table $\mathcal{A}$ where we store the orderings we have already computed, along with the tuple identifiers in trace table. The auxiliary table is clustered on the tuple-identifiers so that the join on tuple-identifiers between computed relation and auxiliary table does not require sorting of auxiliary table. Thus, the cost of computing canonical ordering reduces to the I/O cost involved in scan of auxiliary tables as opposed to sorting some partition.

As we want to ensure that the cost of scanning $\mathcal{A}$ is less than the sorting cost, we make sure that the table $\mathcal{A}$ is smaller in size. This is done by creating new auxiliary table for each canonical ordering. This table consists only of tuple-id and the column with the ordering. The auxiliary tables though come at the cost of space. There is a trade-off between space allocated for auxiliary tables and the cost of transformation. In the best case, the cost of composite transformation is the cost of scanning auxiliary tables when all the orderings had been computed previously and stored. In the worst case, the cost is equal to the cost obtained using the best partitions when either the orderings were not computed beforehand or there was not sufficient space for auxiliary table. In any case, the cost is smaller than the naive computation. In Figure 6(c), the graph shows that



(a)

(b)

(c)

Figure 6: Experiments on traces of 20 million to 52 million rows. (a) The execution time increases linearly with the number of O operators in the query; (b) The execution time vs number of partitions graph for different data sizes. The query consisted of six rank operators. (c) Execution time for a query as the number of ranks already available in auxiliary tables increases.

the execution time of query reduces with increasing memoization i.e. increase in availability of pre-computed ranks. The execution times are given for the best 2-partition strategy for the query.

6.4 Row Store vs Column Store databases

As described earlier, the implementation in row-store databases requires the creation of vertical partitions. If the required partitions are not present, the transformation cannot be done in an optimal manner. In case of column stores, the need for vertical partitions is obviated as it reads only the columns required by the query. Thus, the optimization in column store databases can be done using query rewriting similar to row-store implementation but with the use the original table instead of the partitions. Hence, transformations can always be applied optimally in column stores without any space overhead for the partitions. In theory, the column-store databases look promising but the current implementations of column-store databases could not execute our queries. We suspect that it is due to the nature of queries where the result set has number of records equal to that in the input table. The big size of result set leads to the memory management problems in the column-store database implementations.

7. RELATED WORK

The framework described in this paper was first proposed in [14]. This work described the motivation for the framework and presented the set of operators for transformation. In an extension to this work, we describe the techniques that allow the trace owner to reason formally about the utility and the privacy of the communication traces. In addition, our work describes way to evaluate collusion, privacy and the techniques involved in efficient application of transformations in the system.

K-anonymity [15, 17] and variants [9] apply to the case where there is exactly one record per entity in the data and where there is well-defined fixed-size set of attributes that act as quasi-identifying attributes. Terrovitis et al., have extended the definition of k-anonymity for the privacy-preserving publication of set-valued data containing multiple entries for the same entity [18]. Their work is motivated by market basket data and the utility goals were related to learning association rules. The communication traces in our problem are set-valued in nature. However, communication trace anonymization is significantly different because conventional analysis cannot depend on subtle ordering and correlation among entries. Verykios et al [19] considered the privacy of transactional data in the context of data-mining where they wanted to prevent subsets of association rules from being learned. This was done using deletion or addition of items from the data. This is dependent on the data mining algorithm and is not applicable to our problem.

Differential privacy is a rigorous privacy definition in which noise is calibrated to the sensitivity of queries over a database [6]. Because sensitive entities are described by many tuples in a trace, the differential notion of privacy does not map easily to communication traces. In addition, trace analysts run complex programs over their data and like to have the data in their possession, rather than ask for individual query answers from a server. Although some differentially private schemes can generate sets of statistics that approach synthetic data sets [4], it's not clear how to adapt them to communication traces in a manner that will support common analyses.

For the special case of network traces, anonymization has received special attention by researchers, with IP packet traces the most common case. Proposed anonymization techniques include *tcpurify* [3], the *tcpdpriv* [2] and Crypto-PAn [7] tools (which can preserve prefix relationships among anonymized IP addresses), as well as frameworks for defining transformations, such as *tcpmkpub*. In [2], Xu et al proposed a cryptography based prefix-preserving anonymization scheme for the ip-addresses. This scheme does not transform any other field and is known to be insecure. In [12], Pang et al studied the various fields in traces in some detail and proposed a framework to support anonymization of different fields. In theory, it can publish multiple views but it lacks the tools for analyzing utility of the different views.

Slagell et al [16] recognized the need for a framework to support the trade-off between the security and utility of traces and provide multiple levels of anonymization. In [11], Mogul et al propose a framework that requires an analyst to write the analysis program in the language supported by framework. This program is then reviewed by experts for any privacy or security issues. The approved program is executed by the trace owner and results are provided to the analyst. In our framework, the analyst submits a set of utility constraints and not a general purpose program. Therefore, trace owner does not have the significant burden of evaluating the safety of a program and the owner is not required to provide computational resources to run the program, and allows the analyst to refine their computations on trace.

The framework proposed in [10] tries to achieve balance between utility and privacy by allowing the analyst to submit secure queries to trace owner in a language provided by the framework. These queries permit only certain operations on the fields. The result of the queries consist of aggregate information like counts and histograms and not record level information. This approach prevents the analyst from running a general program which may require information other than aggregate results. Also, by restricting operations, this approach may miss the opportunity to release a trace which may otherwise be safe.

The PREDICT [1] repository has been established to make network traces available for research. The access to respository is authorized only after the purpose and identity of researchers is reviewed and verified. To the best of our knowledge, the anonymization of traces is not customized to the needs analysts and multiple versions of traces are not published.

8. CONCLUSION

We have described a publication framework that allows a trace owner to customize published traces in order to minimize information disclosure while provably meeting the utility of analysts. Using our framework, the trace owner can verify a number of useful privacy and utility properties statically. Such properties hold for all possible traces, and can be verified efficiently. However, some aspects of trace privacy must be evaluated on each particular trace. We have implemented our techniques and quantified trace privacy under example transformations. Also, we have described a dynamic evaluation technique to assess the collusion risks when multiple views of the same trace are released.

In our implementation, we have described ways to apply transformations efficiently to the trace. We have shown that the cost of applying transformation scales linearly with the size of the trace. With the use of memoization, we have been able to show that the cost of applying new successive transformation decreases with time. In future work we would like to implement a trace transformation infrastructure to efficiently support multiple transformations of a trace using parallel data access.

9. REFERENCES

[1] Predict. *https://www.predict.org/*.

[2] Tcpdpriv. *http://ita.eee.lbl.gov/html/contrib/tcpdpriv.html*.

[3] Tcpurify. *http://irg.cs.ohiou.edu/ eblanton/tcpurify*.

[4] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Principles of Database Systems (PODS)*, pages 273–282, 2007.

[5] K. Claffy. Ten things lawyers should know about internet research. *http://www.caida.org/publications/papers/2008/lawyers_top_ten/*.

[6] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[7] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon. Prefix-preserving ip address anonymization: measurement-based security evaluation and a new cryptography-based scheme. *Comput. Netw.*, 46(2):253–272, 2004.

[8] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring tcp connection characteristics through passive measurements. In *Proceedings of INFOCOMM*, 2004.

[9] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.

[10] J. Mirkovic. Privacy-safe network trace sharing via secure queries. In *NDA '08: Proceedings of the 1st ACM workshop on Network data anonymization*, pages 3–10, 2008.

[11] J. C. Mogul and M. Arlitt. Sc2d: an alternative to trace anonymization. In *MineNet '06: Workshop on Mining Network Data*, pages 323–328, New York, NY, USA, 2006.

[12] R. Pang, M. Allman, V. Paxon, and J. Lee. The devil and packet trace anonymization. *ACM SIGCOMM Computer Communication Review*, 36(1):29–38, January 2006.

[13] A. Parate and G. Miklau. A framework for utility-driven network trace anonymization. *Umass Computer Science Technical Report 2008-038*, 2008.

[14] A. Parate and G. Miklau. A framework for safely publishing communication traces. *To appear in CIKM*, 2009.

[15] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. on Knowl. and Data Eng.*, 13(6):1010–1027, 2001.

[16] A. Slagell and W. Yurcik. Sharing computer network logs for security and privacy: A motivation for new methodologies of anonymization. *IEEE/CREATENET SecureComm*, pages 80–89, 2005.

[17] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[18] M. Terrovitis, N. Mamoulis, and P. Kalnis. Privacy-preserving anonymization of set-valued data. *Proc. VLDB Endow.*, 1(1):115–125, 2008.

[19] V. Verykios, A. K. Elmagarmid, E. Bertino, Y. Saygin, and E. Dasseni. Association rule hiding. *IEEE Transactions on Knowledge and Data Engineering*, 16:434–447, 2003.