SC2D: An Alternative to Trace Anonymization

Jeffrey C. Mogul
HP Labs
Palo Alto, CA 94304
Jeff.Mogul@hp.com

Martin Arlitt
HP Labs/University of Calgary
Palo Alto, CA 94304
Martin.Arlitt@hp.com

ABSTRACT

Progress in networking research depends crucially on applying novel analysis tools to real-world traces of network activity. This often conflicts with privacy and security requirements; many raw network traces include information that should never be revealed to others.

The traditional resolution of this dilemma uses trace anonymization to remove secret information from traces, theoretically leaving enough information for research purposes while protecting privacy and security. However, trace anonymization can have both technical and non-technical drawbacks.

We propose an alternative to trace-to-trace transformation that operates at a different level of abstraction. Since the ultimate goal is to transform raw traces into research results, we say: cut out the middle step. We propose a model for shipping flexible analysis code to the data, rather than vice versa. Our model aims to support independent, expert, prior review of analysis code. We propose a system design using layered abstraction to provide both ease of use, and ease of verification of privacy and security properties. The system would provide pre-approved modules for common analysis functions. We hope our approach could significantly increase the willingness of trace owners to share their data with researchers. We have loosely prototyped this approach in previously published research.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Operations

Keywords

trace anonymization

1. INTRODUCTION

Progress in networking research depends crucially on applying novel analysis tools to real-world traces of network activity. Without measurements of the actual behavior of real-world network users, we risk developing models that are either oversimplified, or simply wrong. Implementors need real-world measurements to drive decisions such as the right choice of route-lookup algorithm and the right amount of buffer memory. Network activity traces, made at various layers from packets to user-application interactions, often are the best source of raw measurement data.

Unfortunately, researchers often depend on others, such as ISPs, corporations, and universities, to provide traces. A researcher in organization A might need traces that can only be made at trace-owner organizations B, C, and D. This need can conflict with the privacy and security requirements of the trace-owner organizations. Many raw network traces include information that should never be revealed to others, including personal identify information, secrets such as credit card numbers, traffic patterns that could be analyzed to determine corporate strategy, clues to system vulnerabilities, etc.

The traditional resolution of this dilemma uses *trace anonymization* to remove secret information from traces.[1] Trace anonymization transforms an input trace into an output trace, with the aim of balancing the information needs of a researcher with the privacy and security requirements of the trace owner.

While trace anonymization can often resolve the research-value-vs-secrecy dilemma for certain pairings of research goal and information protection requirements, there are many cases where no satisfactory tradeoff is possible. For example, the researcher might want to know:

- the potential hit rate of a route-lookup cache, while the data-owning organization (such as an ISP) does not want to reveal anything about the structure of its internal network.
- the distribution of the number of different PCs that a distinct person uses during the course of a day.
- how often users accidentally send strings resembling credit card numbers and US Social Security numbers without encrypting them.

For some of these examples, to be sure, it is plausible to construct a transformation on the data that appears to preserve the research-value-vs-secrecy tradeoff, but it can be tricky to get this transformation right. For example, consider a researcher who wants to know the overall distribution of response sizes at a public Web server, and a trace owner who wants to conceal the frequency of access to specific files on the server. Even a trace consisting solely of response lengths might reveal too much: one could crawl the server to discover (size, filename) bindings. Adding *significant* random noise to the sizes in a trace still does not entirely avoid leakage of filenames [18]. In short, any given "anonymizing" transformation can *potentially* leak information if the underlying data has unexpected properties.

[1] Although we follow common practice in using the term *anonymization*, we assume that the privacy and security concerns with traces go beyond simple anonymity.

SIGCOMM'06 Workshops September 11-15, 2006, Pisa, Italy.

It is not always possible to construct a trace-to-trace transformation that fully satisfies both researcher needs and the secrecy constraints of a trace owner. The usual solution is to resort to legally binding agreements combined with trust-building procedures, so that a nervous trace owner is willing to share a trace with a carefully chosen researcher, who promises not to reveal secrets and who can be trusted to do so. Agreements and trust-building involve lengthy negotiations, and often these negotiations fail.

We argue that in such scenarios, trace-to-trace transformation is the wrong paradigm because it operates at the wrong level of abstraction. Rather than focus on providing security and privacy at an intermediate step, we instead focus on the end-to-end problem of generating research *results* that preserve security and privacy.

We propose SC2D, a framework for shipping flexible analysis code to the data, rather than vice versa. Our system design uses layered abstraction to provide both ease of use, and ease of verification of privacy and security properties. The system would provide pre-approved modules for common analysis functions. Sec. 3 describes this design in detail. Although we have not implemented the proposed framework, Sec. 4 describes how we have loosely prototyped this approach in conducting previously published research.

This is an ambitious proposal and we offer it expecting that some aspects might prove too difficult or expensive. An "SC2D-light" design might provide many benefits without as much complexity.

The use of real-world network traces in research is inherently a social and legal problem. Our goal is to respect these societal constraints. *We do not attempt to eliminate the societal conflicts*; our technical approach is designed to support social processes that minimize these conflicts. We aim to change the terms of the trust negotiation, not to eliminate it. One can view this as a form of the "tussle space design" suggested by Clark *et al.* [3].

2. RELATED WORK ON TRACE ANONYMIZATION

Many trace-based research studies have been published using anonymized traces. The community has developed a broad set of anonymization techniques, as well as methodologies to evaluate their impact both on research feasibility and on data privacy and security. For space reasons, we discuss only a few relevant papers; see [4, Ch. 8] for a full treatment of anonymization.

Even the relatively narrow issue of how to anonymize IP addresses while preserving prefix relationships (a requirement for research into route-lookup performance, routing performance, etc.) has proved difficult in practice. Fan *et al.* [7] describe a cryptography-based scheme, but point out that even their scheme is potentially vulnerable to certain attacks.

Pang *et al.* [12] provide an overview of tools that have been designed for trace-to-trace anonymization, and conclude that "anonymization ... is about managing risk." They point out numerous subtle risks in verifying that trace-to-trace anonymizations do not leak, and describe `tcpmkpub`, a general trace-to-trace anonymization framework tool that supports "a wide range of policy decisions and protocols." Much careful work has gone into `tcpmkpub` to prevent leakage, but Pang *et al.* point out that more work remains.

Fahmy and Tan [6] observe that "fill-in" systems that transform well-formed flows into "anonymized" well-formed flows, such as in [13], might not preserve the non-well-formed flows (e.g., attack packets) highly relevant to some intrusion-detection analyses.

3. OUR PROPOSED ALTERNATIVE

In the traditional trace anonymization model, we start by getting the data-owning organization, such as an ISP or corporation, to collect a raw trace at the appropriate point. (This step itself is often fraught with logistical and social issues, but we assume those apply in any approach.) The trace owner then decides to apply an anonymizing transformation, either in consultation with a specific researcher, or with the intention of making the anonymized trace generally useful. Finally, the anonymized trace is shipped to one or more researchers; this step can introduce logistical problems if the trace is large. For example, one of us (Arlitt) has ca. 5 TBytes of trace data, which would be hard to store at many research sites, let alone transmit. Pang and Paxson [13] report capturing 50 GBytes/day at LBNL.

In our SC2D model, we also start with a raw trace. However, in SC2D, the researcher sends an analysis program to the trace owner. The trace owner then runs this analysis program within a carefully-designed framework, and returns the results to the researcher. Alternatively, the trace owner might speculatively run a set of standard analysis programs and publish the results, without a specific researcher's request.

Of course, our approach only works if the analysis program can be trusted not to reveal secrets in the results. We propose a layered solution to this problem:

- **A standardized, safe execution framework**: We can factor out most of the code in any trace analysis into a set of standard functions, with well-defined behaviors. This framework can be distributed as Open Source software, with cryptographic signatures to avoid tampering, and can be security-reviewed by independent experts. Sec. 3.1 describes our proposed framework design in more detail.
- **Interpreted, source-code analysis modules**: Research-specific analysis would be defined at a relatively high level of abstraction by analysis modules, written in a domain-specific interpreted language defined by the framework. Although researchers would have to convince trace owners that these modules do not reveal secrets in their results, the use of a high-level language should simplify the required code reviews. We assume that it can also be designed to provide the same kind of safety and sandboxing guarantees as provided by languages such as Java.

  The analysis modules would be allowed to export results only via constrained interfaces, and raw or intermediate traces would never be allowed to leak out. (It might be possible to apply some results in the design of multi-level secure systems [10], also known sometimes as "taint analysis.")
- **Independent expert review of framework and of analysis modules**: We assume that trace owners would not trust individual researchers to certify the safety of their analysis modules, and would not trust their own abilities to spot problems. Instead, we assume that the community as a whole would support a process of independent expert reviews, somewhat of a cross between the peer-review process for publications and the financial-audit process. The same kind of review process would apply to the implementation of the underlying framework. Sec. 3.2 further discusses the review process.

We see several benefits of our approach:

- **Transparency**: In the traditional trace-anonymization model, it can be hard to tell whether an "anonymized" trace still provides the ability to extract information that should have been secret. By reducing traces to concise research results before anything leaves the hands of the trace owner, we can severely limit the possibility of intentional or accidental breaches of security and privacy. Researchers might still have to justify their need for

specific results by explaining in detail what they mean, but since this is a normal part of any research publication, we do not see it as a burden.

SC2D does not eliminate the trace owner's burden of deciding whether a researcher's proposed analysis reveals too much. However, SC2D turns this into a question solely of whether the research results reveal too much, not whether a trace does.

- **Flexibility for research**: As discussed in Sec. 1, it can be difficult or impossible to sufficiently anonymize a trace without losing information that would enable or improve a research project. By shipping analysis code to the data, we believe we can provide potentially unlimited research flexibility. Also, by providing a standard framework with a high-level language that supports trace analysis, we greatly simplify the process of writing analysis tools (see Sec. 4 for our experience in this respect).
- **No need to ship large traces**: Because traces are never shipped, only results, the logistical issues of shipping large data sets, especially across firewalls, simply disappear.
- **The potential for on-line analysis**: Some organizations prohibit even internal storage of raw traces [12]; SC2D can obviate such storage by performing analysis as data is generated, and then discarding the raw data.
- **Outsourcing of security reviews to experts**: In almost any two-party negotiation, the easiest way to establish trust is to involve a neutral, expert third party. This is especially important when the party with secrets to protect is not expert in security issues. We believe that a crucial aspect of our approach is that it provides a well-defined way to include independent security experts.

Note that we do not propose a model in which an unknown ostensible "researcher" can send code to a trace owner and expect to receive results. We aim to enable researchers and trace owners who already have established some level of trust to increase their trust level.

3.1 Framework design

Our approach depends on a standard framework system, which should provide:

- support for various kinds of traces, including packet traces, routing-protocol event traces, HTTP message traces, NetFlow traces, etc.
- a high-level interpreted language, specialized for the problem of network trace analysis.
- built-in modules for commonly-used functions.
- traditional anonymization transformations, as a "firewall" against unrecognized flaws in analysis modules.

Since our approach places the analysis at the trace owner's site, this effectively forces us to support a high degree of automation, to minimize the logistical burden. This motivates two other features, which would be useful for any trace-based research:

- a trace-handling sub-system, to eliminate the burden on the trace owner to deal with identifying and preprocessing trace files.
- a scriptable experiment-manager subsystem, to eliminate the burden on the trace owner of running multiple analyses with different parameter values, and to manage resources consumed during the experiments.

We discuss each of these points in more detail.

3.1.1 Language design

The design of the interpreted language is a key issue in our approach. We have been strongly influenced by our experience with Bro, a similar framework designed for intrusion detection systems [14]. Bro provides a modular scripting language designed to support analysis of IP network event streams, but can also be used off-line. Our proposed framework would also need to support module composition and re-use, and, like Bro, would need primitives specific to the networking domain. The language should provide safety and sandboxing properties, as does Java, and should be biased in favor of readability to support security reviews (see Sec. 3.2).

Pang and Paxson [13] describe an extension to Bro for packet trace anonymization and transformation. Their system offers many features that would be useful in an analysis language; their language explicitly supports anonymization policies. They observe that the language should make it easy to examine a module for privacy leaks.

Kohler [9] has shown how the Click modular router framework conveniently supports measurement applications. SC2D could borrow Click's approach for specifying the connections between analysis modules, in a way that limits the damage they can do and thus the effort required to review them.

3.1.2 Built-in modules for common functions

Based on our past experience, we believe that a trace analysis framework must include modules for

- statistical analyses; for example, the R language and environment for statistical computing [15], or something like it (such as NNstat [2]). This should support standard representations for things like histograms, CDFs, and PDFs, that can become inputs for further processing.
- a minimal database, such as BerkeleyDB [17], for managing auxiliary data, such as parameters, identity mappings, and other intermediate structures.

Other standard functions will probably prove useful.

3.1.3 Standardized trace formats

In order for SC2D to support the reuse of analysis modules, and the composition of multiple modules written by different researchers, it should provide standardized trace formats, as well as libraries of methods to manipulate them. This standardization should also reduce the cognitive load on experts reviewing the modules for secrecy issues.

Since SC2D is intended to support trace analysis at multiple levels, it will require multiple standard formats (e.g., packet traces, routing-protocol event traces, HTTP message traces, etc.). The trace formats should cover not only the per-event record formats, but also per-trace meta-data such as location and time where the trace was gathered, configuration information such as the filters that were employed during trace gathering, and statistical information such as the number of packets, number of known errors, etc. While some of the statistical information could be reconstructed by reading the whole trace, it might be far more efficient to have this available for quick inspection during later analysis.

Because trace-collection technologies vary widely, and should be outside the scope of the framework *per se*, we will also need a collection of trace converter plug-ins, to translate from other trace formats to those used by SC2D. The framework should sandbox these plug-ins so that they cannot leak information via covert channels, and thus do not themselves need to be certified.

3.1.4 "Firewall" transformations

We usually prefer "security in depth" over designs that place all of the security burden on one, possibly buggy, component. This suggests that the framework should support a set of traditional trace-to-trace anonymization transformations, to be applied before (or perhaps after) other secrecy-preserving techniques. As with other SC2D software, these would be certified and signed by ex-

pert reviewers.

Transformations would be selected based on the specific goals of a research project, but because they would not bear the entire burden of preserving privacy and security, they need not be as draconian as those in a traditional trace-anonymization approach. They could still improve the confidence level of trace owners who do not fully trust either the expert review process or that the framework's implementation is bug-free.

3.1.5 Trace handling sub-system

Much of the effort involved in doing trace-based research is the management of large amounts of trace data. Typical experiments often involve multiple input traces, upper-level traces synthesized by transformation tools, other intermediate processing steps, quality control, etc. It is one thing for researchers to do this tedious and error-prone work themselves; it would be hard to convince trace-owners to do this work manually as a consequence of the SC2D approach. Therefore, the framework must make trace handling as simple and labor-free as possible.

A trace-handling sub-system (THSS) should support:

- **The use and merging of multiple traces**: Quite often, a single analysis will require multiple input traces. For example, it might be necessary to capture input and output packets, or packets from different ISPs, at different monitors, or it might be necessary to break a long trace into multiple serial sub-traces in order to avoid file-size limits (we encountered both issues in previous work [1]). The THSS should be able to merge such multiple traces into a unified stream.

  In other cases, it might be necessary to capture traces at multiple sites (e.g., to measure wide-area networking effects), thus getting multiple views of the same events. The THSS should be able to *reconcile* such traces into a unified stream (see [16] for a discussion of this approach).
- **Trace-to-trace anonymization modules**: as described in Sec. 3.1.4.
- **Trace quality cleanup**: Real traces are full of bogus events. This is true especially for high-level traces synthesized from packet-level traces, which may suffer from missing or re-ordered packets, or simply from unexpected behavior. Traces can also suffer from *end effects*, since a trace might start or end in the middle of a connection. The THSS should provide mechanisms for detecting, counting, and deleting bogus events. (We do not say this is easy, and successful deletion of bogus events runs the risk of biasing the subsequent results.)
- **Timestamp correction**: Traces made at multiple sites may suffer from clock skew, which can interfere with timing analysis or cause mis-ordering of events. The THSS should provide mechanisms for detecting clock skews and correcting event timestamps.
- **Slicing**: Sometimes the analysis only applies to a particular slice of a trace. The THSS should support slicing by time period, host or network IDs, protocol, event type, etc.
- **Meta-data tracking**: The THSS should track trace meta-data as described in Sec. 3.1.3, and provide viewing and searching facilities for this meta-data.

3.1.6 Experiment manager sub-system

Most research projects involve conducting multiple experiments. For example, one might want to simulate several caching algorithms, each with several parameter choices, against several traces. As with trace handling, the SC2D approach risks shifting this burden to the trace owner. In our experience, many errors can creep into this phase of a research project, so automation is essential.

The trace handling framework should include a scriptable experiment manager (EM) that can stage multiple experiments, properly keeping track of which results came from which experiments. The EM should be able to exploit parallel resources where possible, without violating data dependencies and without overloading the resources provided by the trace owner. The EM should recover automatically from experiments aborted due to failures or resource constraints.

The EM must also enforce the distinction between "results" that are OK to release to researchers, and all other data, which must be treated as private.

3.2 Expert review process

Our approach critically depends on the successful use of an independent expert review process to certify the security and privacy properties, both of the framework and of the analysis modules. This is both a technical problem and a social problem.

The technical issues include:

- **Careful language design**: The design of the interpreted analysis-module language will affect how easy it is to determine if modules have security bugs.
- **Verifiability of the framework implementation**: Most of the code will be in the framework implementation, not the analysis modules, and this framework will be responsible for enforcing the assumptions underlying the analysis-module review. The framework code must therefore be as transparent as possible.
- **Review of composed analyses**: Research results will be produced by the composition of a set of analysis modules, and so a security review will have to review the global behavior of the entire set, not just the individual pieces.

  It might be useful to support *proof-carrying code* (PCC) mechanisms [11] or taint analysis, as a way for researchers to make formal assertions about what an entire analysis does *not* do. For example, PCC can prove that a module does not access data except as specified in its interface definition. Taint analysis can prove that the output of a module does not depend on privacy-sensitive input data.
- **Signing mechanisms**: Once the framework and analysis modules have been reviewed, they should be cryptographically signed, with traceable authentication, so that trace owners can be sure they are getting properly-reviewed code.
- **Automatic leakage detection**: Either the expert reviewers or the trace owner might wish to augment the review process with heuristic-based techniques, such as described by Pang *et al.* [12], to check for privacy leaks (e.g., checking for patterns typical of credit card or social-security numbers).

The social issues include:

- **Choice of experts**: We will need to find security experts with appropriate skills and trustworthiness.
- **Funding model**: Security experts might not be willing to work under a zero-funds model akin to the peer review mechanism, since they might not be benefitting from a symmetrical exchange of work. The networking research community could ask funding agencies to sponsor the review process, but this issue could be the achilles heel of the entire approach.
- **Detection of cheating**: Even the most honest review process could be subverted. We might need some sort of auditing process (both technical and human-based) to look for attempts to spy via ostensibly "research-only" analysis modules. Audit support might also increase the confidence of trace owners.

  Ideally, we might hope for formal proofs of the privacy and security properties of an analysis, but we doubt this will be feas-

ible soon, especially because it might be hard to formally specify the precise properties. We suggest a "many eyeballs" approach is at least superior to current alternatives.

- **Pre-publication confidentiality**: The review process forces researchers to reveal their hypotheses and techniques long before the research is ready to publish. As with the peer-review process for papers, the expert review process might require a pledge of confidentiality to researchers who submit modules for review.
- **Liability**: If an analysis module reviewed and cleared by "experts" turns out to have a privacy bug, can these experts be sued? If so, would anyone be willing to serve as an expert? If not, would data owners trust the process? It might be that the community would indeed trust a "best effort" expert-review model, as this is more security checking than almost all commercial software undergoes today.

We note that US law requires Institutional Review Boards (IRBs) to do prior review of the use of human subjects in federally-funded research [5]. It would not be a big stretch to see the expert review of trace analysis modules as analogous to these IRBs, since the problems of data privacy in traces intersect with other aspects of the use of humans as research subjects. Our community might learn something from the experience of IRBs.

4. EXPERIENCE WITH A PROTOTYPE

In previous work [1], we reported trace-based experiments to validate approaches to predict the latency of short TCP transfers. For that project, each researcher was the "owner" of a trace that could not be shared directly. In theory, we could have used traditional anonymization, but we knew of no pre-existing tool that preserved all the data we needed, including TCP-level RTT measurements, HTTP-level data transfer timing and byte counts (one TCP connection can carry many HTTP messages), and HTTP request types and status codes.

Out of necessity rather than design, we developed a simplistic SC2D approach to this project. We used a Bro script to convert raw packet traces to HTTP-level traces with the necessary fields (this functionality might be a useful "standard module"), then used a combination of *R* and *awk* scripts to generate research results, all held together with shell scripts. Only the results left the trace owner's site, so we did no actual trace anonymization.

The high-level constructs in Bro meant that our Bro scripts were relatively simple (800 lines for the primary script; see *http://bro-ids.org/bro-contrib/network-analysis/akm-imc05/* for our software).

Our experience suffered from our *ad hoc* approach to managing the workflow, which involved multiple steps and no tracking tools. The experiment scripts had to be parameterized for each site, and we sometimes got confused about which experiments had to be rerun after a script or parameter change. We also had some trouble managing CPU resources for long-running experiments, as well as in managing disk space. The THSS and experiment manager proposed in Secs. 3.1.5 and 3.1.6 were motivated by these problems.

While this project served as motivation for SC2D, it was not a true prototype. Our project involved three people who have known each other for over a decade, and both sides of the "researcher vs. trace-owner" negotiations were, in fact, researchers. Therefore, we did no actual code review; we simply elected to trust each other, and we shared an informal understanding of what the results (statistical summaries and graphs) revealed. (Note that we trusted each other's code, but were not allowed to trust each other with direct access to the raw data; these are two different kinds of trust.)

5. POTENTIAL DRAWBACKS

In this section, we briefly discuss some potential drawbacks of our approach. Space prevents a full treatment, nor do we currently have solutions for all of them. We note that most of these, while challenging technical or social problems, are merely *hard* to solve, while the tradeoff between trace anonymization and data utility can be *impossible* to solve in some cases.

Debugging the analysis software will probably be much harder, as bugs can arise that might not be revealed during testing on the developer's own data. Each revision of an analysis module would presumably have to be resubmitted for expert review before being tested against private data, since a "simple bug fix" could introduce novel vulnerabilities. However, technologies such as PCC or taint analysis might sometimes allow automatic proofs that minor bug-fixes do not change the security and privacy properties of a module; certainly, one could expect these techniques to make re-review easier.

Debugging of trace analyses often involves solving puzzles: the results are unexpected in some strange way. We often solve such puzzles by exploring the underlying data in minute detail; this would be a lot more challenging using SC2D, unless the data owner is an active participant.

Longevity of data could be less assured. With trace anonymization, researchers (or sometimes community archives) can hold the traces as long as necessary for purposes such as reproducing or verifying results. With SC2D, data owners might have less incentive than researchers to keep large data sets around, or to make sufficient backups. On the other hand, the potential to run SC2D in an online mode means that data owners with policies against any storage of raw traces might still be able to cooperate with researchers.

One should also not assume that replication of a research result requires the use of the *same* trace. In fact, given that any particular trace is likely to be atypical in some aspects, the generality of trace-based research results ought to be proved using multiple traces from different sites.

Serendipity is less likely, since analyses will be chosen in service of specific research goals rather than random exploration. The goal of SC2D is to avoid revealing more information than necessary to meet the stated research goals, so in some sense the approach is inherently anti-serendipitous.

Analysis across multiple sites could be much harder using SC2D. Such analyses often involve tracking whether the same event or data appears at multiple sites, which could be in direct conflict with data-owner privacy policies (especially for mutually distrusting sites). Perhaps zero-knowledge proof techniques [8] could be applied, although these are likely to be expensive.

Covert channels are probably impossible to eliminate entirely. SC2D, through both technical means and the expert review process, might be able to at least quantify the bandwidth of the channels that remain.

Incentives for data owners to participate are not clear. SC2D shifts several burdens from researchers to data owners, including trace storage and computational resources. We note, however, that many data owners have been willing to support trace-based research, either through altruism or because they expect the research results to benefit them in the long run.

The owners of a popular data set might have to deal with multiple researchers competing for analysis resources just before a deadline. In one sense, this represents a success (as it implies the high value of the data), but it could also be a headache. The THSS might need to support resource-reservation mechanisms, which would also be useful if the data owner is providing the analysis resources from a

pool of systems that can also have higher-priority uses.

6. SUMMARY

SC2D could create new opportunities for trace owners and researchers to work together. The design has many potential limitations and risks, which would take another six pages to describe. We hope that our proposal leads, at least, to a productive discussion.

7. ACKNOWLEDGMENTS

We thank Sonia Fahmy, Terence Kelly, Greg Minshall, Vern Paxson (particularly for Sec. 5), and especially Balachander Krishnamurthy, whose critique of early drafts of this paper helped us focus and sharpen our arguments.

8. REFERENCES

[1] M. Arlitt, B. Krishnamurthy, and J. C. Mogul. Predicting short-transfer latency from TCP arcana: A trace-based validation. In *Proc. Internet Measurement Conference*, pages 213–226, Berkeley, CA, Oct 2005.

[2] R. T. Braden. A pseudo-machine for packet monitoring and statistics. In *Proc. SIGCOMM*, pages 200–209, Stanford, CA, Aug. 1988.

[3] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in Cyberspace: Defining Tomorrow's Internet. In *Proc. SIGCOMM*, pages 347–356, Pittsburgh, PA, Aug 2002.

[4] M. Crovella and B. Krishnamurthy. *Internet Measurements: Infrastructure, Traffic and Applications*. John Wiley and Sons Ltd., Chichester, UK, 2006.

[5] Dept. of Health and Human Services. CFR Title 45 Part 46: Protection of Human Subjects. `http://www.hhs.gov/ohrp/humansubjects/guidance/45cfr46.htm`, 2005.

[6] S. Fahmy and C. Tan. Balancing Privacy and Fidelity in Packet Traces for Security Evaluation. Tech. Rep. CSD-04-034, Purdue Univ., Dec. 2004.

[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. *Computer Networks*, 46(2):253–272, Oct. 2004.

[8] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th Symp. on the Theory of Computation*, pages 291–304, Providence, RI, May 1985.

[9] E. Kohler. Click for measurement. Technical Report TR060010, Dept. of Comp. Sci., UCLA, Feb. 2006.

[10] A. C. Myers and B. Liskov. A Decentralized Model for Information Flow Control. In *Proc. SOSP*, pages 129–142, St.-Malo, France, Oct. 1997.

[11] G. C. Necula. Proof-Carrying Code. In *Proc. POPL*, pages 106–119, Paris, France, Jan. 1997.

[12] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. *SIGCOMM Comput. Commun. Rev.*, 36(1):29–38, 2006.

[13] R. Pang and V. Paxson. A High-level Programming Environment for Packet Trace Anonymization and Transformation. In *Proc. SIGCOMM*, pages 339–351, Karlsruhe, Germany, Aug. 2003.

[14] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, Dec. 1999.

[15] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2003. ISBN 3-900051-00-3, `http://www.R-project.org`.

[16] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, C. Killian, and A. Vahdat. WAP5: Black-box Performance Debugging for Wide-Area Systems. In *Proc. WWW*, Edinburgh, UK, May 2006.

[17] Sleepycat Software Inc. Berkeley DB. `http://www.sleepycat.com/products/bdb.html`.

[18] Q. Sun, D. Simon, Y.-M. Wang, W. Russell, V. Padmanabhan, and L. Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *Proc. IEEE Symp. on Security and Privacy*, pages 19–30, Oakland, CA, May 2002.