Semi-Automated Discovery of Application Session Structure

Jayanthkumar Kannan
UC Berkeley
Berkeley, CA, USA
kjk@cs.berkeley.edu

Jaeyeon Jung
Mazu Networks
Cambridge, MA, USA
jyjung@mazunetworks.com

Vern Paxson
International Computer Science Institute and
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
vern@icir.org

Can Emre Koksal
EPFL
Luasanne, Switzerland
emre.koksal@epfl.ch

ABSTRACT

While the problem of analyzing network traffic at the granularity of individual connections has seen considerable previous work and tool development, understanding traffic at a higher level—the structure of user-initiated *sessions* comprised of *groups* of related connections—remains much less explored. Some types of session structure, such as the coupling between an FTP control connection and the data connections it spawns, have prespecified forms, though the specifications do not guarantee how the forms appear in practice. Other types of sessions, such as a user reading email with a browser, only manifest empirically. Still other sessions might exist without us even knowing of their presence, such as a botnet zombie receiving instructions from its master and proceeding in turn to carry them out.

We present algorithms rooted in the statistics of Poisson processes that can mine a large corpus of network connection logs to extract the apparent structure of application sessions embedded in the connections. Our methods are semi-automated in that we aim to present an analyst with high-quality information (expressed as regular expressions) reflecting different possible abstractions of an application's session structure. We develop and test our methods using traces from a large Internet site, finding diversity in the number of applications that manifest, their different session structures, and the presence of abnormal behavior. Our work has applications to traffic characterization and monitoring, source models for synthesizing network traffic, and anomaly detection.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.3 [**Computer-Communication Networks**]: Network management

General Terms

Measurement, Algorithms

Keywords

Traffic Analysis, Application Sessions, Anomaly Detection

IMC'06, October 25–27, 2006, Rio de Janeiro, Brazil.
Copyright 2006 ACM 1-59593-561-4/06/0010 ...$5.00.

1. INTRODUCTION

Previous studies analyzing network traffic have extensively examined behavior at the scales of individual packets [11, 18, 28, 41] and connections [7, 9, 25, 26, 37, 40]. At a broader scale, however, the structure of *sessions* of related connections involving the same host has seen considerably less investigation, with the main efforts being studies of specific, individual applications such as Web client sessions [1, 33] or FTP user sessions [29]. Yet there is great utility in understanding the nature of sessions because doing so helps provide a foundation for forming *source models*: descriptions of network activity in terms of what a source is attempting to achieve using the network, as opposed to the final result after the source's host software breaks this task down into individual connections that are in turn further shaped, sometimes drastically, by network conditions. The major benefit of source modeling comes from expressing network traffic in terms abstract enough to facilitate exploration (via analysis or simulation) of how the network traffic needed to support a given task might manifest under different conditions and transport mechanisms [10].

We will use the term *session* to denote a group of connections associated with a single network task, where by "task" we mean the activity that emanates from an external event that serves as the *causal origin* of the connections. Thus, a "task" in this context can be quite abstract, such as a user deciding to process their email or conducting an interactive login to a remote host. It will however often be convenient or apt for us to refer to a session using the layer-7 notion of "application" as a proxy for these abstract tasks. For example, we will refer to an "FTP session" rather than a "user transfering files session" because from a network trace perspective what we directly see is an instance of a TCP port 21 connection (associated with the FTP control channel) and perhaps subsequent port 20 or ephemeral port connections (data transfers directed by the control channel).

Using this terminology, we argue that little is presently available in terms of general tools for understanding the *qualitative* structure of application sessions. We believe that such an understanding is useful to researchers and administrators who may benefit from working with higher-level abstractions of network traffic. For researchers, these abstractions promise to aid with traffic characterization and monitoring, constructing source models for synthesizing network traffic, and anomaly detection. For administrators, the hope is to provide them with richer information for framing network policies and gaining insight into how their network is used. Finally, we hope that the semi-automated nature of our approach will help both types of users to keep up with the evolving and diverse landscape of Internet applications.

The aim of our algorithms is to mine a connection-level trace to derive abstract descriptions of the session-structure for the different applications present in the trace. These algorithms work in general terms, without requiring any *a priori* knowledge about a particular type of application. The procedure operates in a *semi-automated* fashion: we aim to provide an analyst with high-quality information reflecting different possible abstractions (*descriptors*) of an application's session structure. We express these descriptors as regular expressions, or, equivalently, deterministic finite automata (DFAs), which capture the order, type, and directionality of the connections that comprise a session, but not their interarrival timing. The different abstractions provided trade off economy-of-expression versus more detailed fidelity to the observed data.

Application sessions manifesting in network traffic can have different basic natures. Some applications have an explicitly defined session structure such as FTP sessions that consist of a FTP control connection followed by a number of data transfer connections. Other sessions reflect a looser structure that arises from how end-user software, or the end user themselves, drives network access in order to perform a task. For example, for sessions beginning with an SMTP connection, we find a plethora of additional activity reflecting the fact that transferring email can be invoked in a variety of contexts: by mail servers (some of which initiate Ident connections back when contacted with incoming mail), by mail clients (such as Thunderbird or Outlook, which may be configured to establish SMTP connections along with POP3/IMAP4 connections for retrieving email), or by users (who may read their mail and browse alongside, leading to HTTP connections). Still other sessions arise from hostile activity, some of which runs counter to the session structures exhibited by benign traffic. While we do not aim to identify these in a comprehensive fashion, we note that our structural inference can provide a baseline for applying anomaly detection techniques for detecting some forms of attacks. See §6.3 and §7 for more details.

Our method for discovering session structures uses as input a connection-level traffic trace. The method operates in two stages: *Session Extraction* and *Structure Abstraction*.

Session Extraction is a statistical algorithm that reduces a stream of connections down to a stream of sessions. We base it on modeling the temporal characteristics of session arrivals, using the important observation that, in comparison with connections from different (and hence independent) sessions, connections belonging to the same session tend to occur "close" to one another. An important limitation of our current work is that we only capture sessions between a single pair of hosts, whereas many forms of sessions (e.g., Web surfing) naturally include a local host's interactions with multiple remote hosts. In the development of our approach, we note at several points ways in which it might be generalized in this fashion.

Structure Abstraction operates on the set of sessions extracted for a given application and attempts to infer succinct session descriptors that capture (at varying levels of abstraction) the structure of sessions typically generated by the application. We aim for this step to provide both economy of description and insight by suggesting apt abstractions. To do so, the inference framework includes a set of generalization rules that simplify or transform the raw descriptions inferred directly from a trace. We present these generalized descriptors to the analyst in terms of *complexity-coverage curves*: the analyst can opt for complex representations for more precise coverage of the set of sessions empirically observed, or for simpler, more abstract representations that may better capture "the heart of the matter". We found that often the curves exhibit inflection point "sweet spots" that mark qualitative transitions between adding complexity and gaining more coverage.

We evaluated our scheme using a month's trace collected at the border of the Lawerence Berkeley National Laboratory, which on average saw about 2,700,000 connections each day. We used the first half of the trace to develop and calibrate our methods, then applying them to infer descriptors for about 40 different applications in the second half of the trace. These include well-known ones such as content-transfer (e.g., SMTP, FTP, HTTP), remote access (e.g., SSH, Telnet, remote exec), database (e.g., OracleSQL, MySQL), peer-to-peer (e.g., BitTorrent), and mapping and authentication (e.g., RPC portmapper, LDAP, Kerberos). We also encountered applications little discussed in the literature: remote desktop (e.g., Timbuktu, Groove), engineering/scientific (e.g., ProEngineer, Legato, GridFTP), and lesser-known peer-to-peer (e.g., KCEasy). When possible, we validated our inferences using published protocol specifications, but for applications whose public documentation does not specify an overall session structure, often we could only assess the plausibility of the inferences based on what we could determine about the application's operation.

The outline for the rest of the paper is as follows. We begin with some background in §2 and discuss related work in §3. We develop our session extraction mechanism in §4 and our structure abstraction mechanism in §5. In §6 we present an evaluation of these mechanisms along with some preliminary results on detecting anomalous traffic. We cast our work in a wider perspective in §7 and conclude in §8.

2. BACKGROUND

We first frame the problem setting by specifying the inputs processed by our algorithms and the terminology we will use in their development.

Traffic Characteristics Inputs. Our algorithm works using connection-level information obtained from passive network monitoring. While in principle our approach could work applied to monitoring inside a site or within a backbone, our development has been in the context of traces recorded at a site's border. For every TCP connection, the monitor records the IP addresses of the local and remote hosts, direction (incoming/outgoing), timing information (start time, duration), and connection status (whether successfully established). One could generate similar information for non-TCP flows, but to date we have evaluated our mechanism only for TCP traffic, using traces of SYN, FIN and RST packets.

Terminology. We denote a connection C by the tuple (*proto, dir, remote-host, local-host, start-time, duration*). *proto* specifies the service associated with the destination port X of the connection (e.g., FTP or HTTP), or, if X does not have a specific service associated with it, we use "*priv-X*" for ports $X < 1024$ (usually only available to privileged processes) and "*other-X*" for $X \geq 1024$. For these latter, in some contexts we will use "*ephemeral*" to indicate that it routinely varies. *dir* takes the value "**in**" or "**out**" indicating the connection was incoming (initiated by the remote host) or outgoing (locally initiated). *remote-host/local-host* is the IP address of the remote/local host, and *start-time* and *duration* denote the beginning times and duration, respectively.

We define the *type* $T(C)$ of a connection C as the tuple (*proto, dir*). We allow some fields to be absent in our tuple notation for connections; the value of such omitted fields will be clear from context.

We define a *session* as a sequence of connections $S = (C_1, C_2, \ldots, C_n)$ that have a common causal origin. In this work, we only consider sessions that involve a single local-host and single remote-host, so implicitly all connections involve only these two hosts. We define the *application*, $A(S)$, associated with a session S as the type of the first connection C_1, i.e., $T(C_1)$. This is imperfect since different applications might manifest with the same type of initial connection, or the same application with different initial connections, but in the absence of ground truth regarding the true user/application sessions in a traffic stream, it strikes us as arguably a reasonable approximation.

We say that a session S *belongs* to the *session type* $ST(S) = (T_1, \ldots, T_n)$, if for all $i \leq n$, $T_i = T(C_i)$. Thus, $A(S)$ categorizes which application a session S belongs to, while $ST(S)$ captures the structure of the entire session in terms of the *proto*'s of each connection and their directionality.

DFA Visualization. In our DFA visualizations (produced by FSA utilities software [12] and Graphviz visualization software [13]), we label the start-state as 0, and distinguish between accepting states and non-accepting states with shading. We label edges with the type of the corresponding connection and a direction tag, "_in" or "_out" indicating incoming/outgoing connections respectively. Finally, we use the thickness of the edge to visually convey how often each edge occurred in the trace.

3. RELATED WORK

The main area of previous work related to our effort consists of studies characterizing network flows at various granularities: packets, connections, and sessions.

Early Internet traffic studies focused primarily on the dynamics of individual packets in terms of comprising flows [4, 14, 15] and with regard to correlational structure [11], especially self-similarity [18]. Subsequent work then characterized the conditions encountered by streams of packets as they traversed the Internet [3, 28, 41]. These studies were accompanied by other early ones that analyzed traffic in terms of connection properties. These included measurements of connection characteristics as seen at different sites [9], dynamics seen within connections [21, 25], transport behavior [27], structural contribution to self-similarity [37], connection performance limitations [40], and numerous characterizations of different applications (e.g., [7, 26] for older ones).

Studies of both packet and connection dynamics have continued to expand into a large body of subsequent work. However, the literature examining application session structure in abstract terms (that is, not simply the structure of a particular application such as Web access or streaming audio/video) has been much more limited. Particularly relevant for our work is Paxson and Floyd's characterization of the connection-level and session-level behavior of applications [29]. This work studied the wide-area TCP arrival processes at differing levels of granularity: packets, connections, and sessions. The authors found that *session* arrivals were generally well-modeled by a Poisson process with hourly rates, even though individual *connection* arrivals were not. More recently, Nuzman *et al.* [24] studied the arrival characteristics of HTTP connections, finding that when these connections are suitably aggregated into sessions, the sessions indeed reflect a Poisson process. These observations form the basis for our Session Extraction algorithm. Characteristics of Web traffic, such as the prevalence of pipelined and persistent HTTP connections, have been studied by analyzing HTTP connections that correspond to requests for a single Web page and its embedded objects [33]. Our work focuses on higher-level patterns of connections (possibly on different ports) exhibited by protocols, such as HTTP, and does not deal with specific application-level characteristics.

In addition, our work is similar to spirit to recent work that aims to automatically infer application-level packet structure. Machine-learning based techniques [19, 22] have been used to identify the application that a particular flow belongs to *without* relying on port numbers; our work is concerned with inferring the application's internal *session structure* and simply uses port numbers to identify applications. Protocol Informatics [31] uses sequence analysis techniques from bioinformatics to identify protocol fields in unknown or poorly documented applications. RolePlayer [8] uses byte-stream alignment algorithms, along with knowledge of a few low-level syntactic conventions, to infer protocol formats to enable RolePlayer to cheaply emulate the application-level behavior of a previously seen client or server. Our work is complementary to these efforts in that we focus on higher-level abstractions of sessions, while they are geared towards more fine-grained characterizations of traffic.

In abstract terms, our work is about discovering and characterizing *causality* in network traffic: which network activities are due to previously seen activities. This theme has been pursued by several lines of work in intrusion detection. First, host-based schemes have related network traffic received by a host with subsequent code executed by the host [5, 6, 23]. In addition, other work has tracked causality in network traffic in terms of an attacker moving among a set of nodes [34], identifying which hosts have infected other hosts [17, 38], and detecting "stepping stones" whereby attackers relay their traffic through previously compromised machines to obscure their identity while attacking other machines [2, 35, 39, 42]. Our work differs from these in that, first, we aim to discover overall patterns of activity rather than detect individual instances; and, second, we do not have to contend with an adversary motivated to thwart our analysis. As a consequence, we can tolerate a greater degree of statistical uncertainty and decision errors. Our statistical test based on a Poisson model of connection arrivals bears resemblance to the stepping stone detection algoritm in Blum *et al.* [2]. [2] presents a test based on modeling packet arrival as a Poisson process, which is then generalized to other distributions as well.

4. SESSION EXTRACTION

Our discovery process begins with *Session Extraction*: reducing a stream of connections to a stream of application-level sessions, where each session comprises a sequence of connections. We first detail different types of session structures (including homogeneous and mixed sessions), then describe how we extract homogeneous sessions, and finally how we extract mixed sessions.

4.1 Types of Sessions

The simplest possible session structure is a lone connection by itself, which we term a *singleton*. Next in complexity comes sessions consisting of consecutive invocations of the same application protocol, all with the same directionality, which we term *homogeneous sessions*. Last in complexity for the types of sessions we tackle in this work are sessions involving different connection types as well as varying directionality, *mixed sessions*. (More complex still are sessions involving multiple remote hosts. Extracting these remains for future work, though we have some preliminary results indicating its likely feasibility [16].)

Different applications vary widely in the prevalence they exhibit for each of these types of session structure. For example, the algorithm we develop classifies 11% of LDAP client sessions as singletons and 88% as homogeneous sessions (with 1% being mixed sessions); for SSH client sessions, 80% singletons vs. 18% homogeneous sessions; and for Grid FTP client sessions, 58% vs. 0% (with 42% being mixed sessions). For about half of the 40+ applications we examined in our trace, simple forms of sessions dominate, while for the rest, sessions often involve somewhat more complex structure. In addition, for some protocols (Web surfing, peer-to-peer) many sessions involve multiple remote hosts, which our present structure abstraction does not aim to capture. Also, as our results later show, for applications that only rarely exhibit mixed sessions, sometimes these are sessions of particular potential interest to an analyst.

4.2 Extracting Homogeneous Sessions

Consider an algorithm that processes a stream of connection arrivals $C_1, \dots$ in an online fashion. On observing a new connection C_i, the algorithm must decide whether: (a) C_i is part of a current session, or (b) C_i represents the beginning of a new session.

We first employ a simple heuristic to identify homogeneous sessions that, as explained before, consist of consecutive invocations of the same application protocol. For example, a user may invoke their mail client to send multiple mails in a single sitting, leading to consecutive SMTP instantiations. The same holds for Web browsing as reported in [24], which shows that one can capture HTTP sessions by simply considering HTTP connections less than a time T_{aggreg} apart as part of the same session. Their work found $\mathsf{T}_{aggreg} \approx 100$ secs as suitable. We generalize this *aggregation rule* as follows. For a connection C_i, if we already have an existing active session $S_j = (C_1^j, \dots, C_n^j)$ between the same pair of hosts and involving the same protocol and direction (i.e., $A(S_j) \equiv T(C_1^j) = T(C_i))$, and for which the most recently seen connection of S_j, C_n^j, arrived less than T_{aggreg} in the past from C_i's arrival, then we consider C_i part of S_j.

4.3 Extracting Mixed Sessions

The aggregation rule does not help for connections either involving different protocols, or somewhat further apart. The former are particularly interesting, as these potentially reflect mixed sessions. In this case we attempt to assess possible *causality*, as follows. If C_i's arrival is indeed part of an ongoing session $S_k = (C_1^k, \dots, C_m^k)$, then we can consider C_i as "triggered" (caused) by the connection C_1^k; C_1^k represents the start of the session S_k, and thus serves as a representative of the event that led to the initiation of the session.

We base our approach on the observation that if C_i is a "triggered connection", i.e., causally related to S_k, then the arrival of C_i is likely to be "closer" to S_k, in comparison to the case where C_i is a "normal connection" (no causal connection to S_k). We now proceed to develop a more formal statement of this intuition by framing the problem in terms of hypothesis testing, and then explain a statistical detection algorithm that has a bounded false positive rate.

Modeling Normal Traffic. We can frame the problem of distinguishing between triggered and normal connections in terms of hypothesis testing [32]. In this formulation, we use the arrival time of a connection to choose between two hypotheses: the *null* hypothesis that the connection is normal, versus the *alternative* hypothesis that the connection is triggered.

Modeling the alternative hypothesis requires capturing the statistical dependence of the arrival time of the triggered connection on the arrival time of the triggering connection, but this may depend on the semantics of the specific application session itself. Fortunately, as we will show shortly, the null hypothesis is easier to model. Therefore our abstraction algorithm discovers likely causality, and hence sessions, by building a model for the arrival characteristics of untriggered (normal) connections. We then identify connections whose arrivals deviate from this model as triggered connections.

At the heart of our approach lies the empirical observation that the arrival of user-initiated sessions is generally well-modeled as a Poisson process, stationary over time scales of an hour [24, 29]. Here, *user-initiated* means sessions instigated by human activity rather than machine activity with a correlational structure (such as periodic daemons). Note that although the model of the arrival process can be relaxed from a Poisson process to a renewal process, the Poisson assumption makes the false positive analysis simpler (as will be seen in Section 4.3).

To fit within this framework, we estimate rates for each type of session using a sliding window of duration $\mathsf{T}_{rate} = 1$ hr. Our model for session arrivals views the activity of local hosts as independent from that of other local hosts, and, further, independent of sessions of other types involving the same local host. We therefore maintain separate notions of session rates for the different types of applications in which each host participates. Finally, our definition of session type also includes directionality, accounting for the difference in the arrival characteristics of clients and servers.

Causality Detection Algorithm. We now describe a statistical test that identifies triggered connections by using our model of normal traffic.

Consider the arrival of two connections C_i, C_j with types denoted T_i, T_j. Assume that C_i is the connection at the start of an ongoing session S_1. Let us provisionally assume that C_j marks the beginning of a new, separate session, S_2. Denote the arrival rates of these session types as λ_1 and λ_2. Let the interarrival time between C_i and C_j be x. Now consider the alternative that C_i triggered C_j. In this case, in general we presume to find x significantly lower than the case when C_i and C_j are unrelated; we base this presumption on the expectation that connections due to a common origin will tend to come somewhat close together. Our strategy is therefore to estimate the probability P of observing an interarrival x for the null hypothesis of the connections being unrelated, and to deduce that C_i triggered C_j, and therefore C_j belongs to S_1, if P is less than a confidence threshold α.

Let T_1 and T_2 be two sessions whose arrivals follow independent Poisson processes with rates λ_1 and λ_2 respectively. Consider the event of an arrival of T_1 followed no later than x seconds by an arrival of T_2. We denote by $P[T_1, T_2, x]$, the expected number of such events per one unit of time. Given such a formulation, our causality detection algorithm proceeds as follows. We first categorize connections into different types; estimate rates for each of these types; and then use these computed rates along with the threshold α to detect triggers. More specifically, on the arrival of a connection C (either incoming or outgoing) of type T involving a local host L, we perform the following actions:

- Let the sessions observed at L in the previous $\mathsf{T}_{trigger}$ seconds be $S_1, S_2, \dots, S_n$, where $\mathsf{T}_{trigger}$ is a threshold specifying the maximum interval that can separate a triggered connection from the most recent activity in the session. In our study, we set $\mathsf{T}_{trigger}$ to be 500 sec.
- If any session S_i (a) has the same type as C, (b) involves the same remote host, and (c) had a connection arrival within a time window T_{aggreg} of C, then we add C to the most recent such S_i, and we are done. This is the simple aggregation heuristic discussed above.
- Estimate the rate of connection arrivals at L for each session type within the past T_{rate} seconds (3600 sec in our study). We form our estimate as simply the average interarrival time between sessions of each type, over the window size T_{rate}.
- For $1 \leq i \leq n$, compute $P[T_i, T, x_i]$, for x_i the interval

between the arrival of S_i and C. Note that at this point C differs in type from S_i.

- If $P[T_i, T, x_i] < \alpha$ and C and S_i involve the same remote host, then add C to S_i. (We conceptually defer the test for the same remote host to this point because when expanding our work to discover sessions involving multiple remote hosts, it is at this point that we will modify the inference algorithm.) Also, note that if $P[T_i, T, x_i] < \alpha$ holds for multiple T_i, then C is added to *all* such sessions S_i.
- If the probability test does not identify C as belonging to any ongoing session, then C is considered to be the first connection of a new session S_{n+1}.

Naturally, the performance of the above algorithm depends critically on the parameter α. Too low a value may mean we miss certain causal links (false negatives); too high may lead to falsely aggregating unrelated connections (false positives). The false negative rate is difficult to characterize analytically due to the lack of a statistical model for the arrival of triggered connections, so we rely on empirical analysis for evaluating it. Our choice for $P[T_1, T_2, x]$, however, allows us to provide a bound on false positives; see below.

False Positives. In this section, we establish an upper bound for $P[T_1, T_2, x]$, where T_1 and T_2 are two different types of sessions, and then use it to upper-bound the false positive rate (i.e., the number of false positives per second) in the presence of M different session types.

THEOREM 1. *Let λ_1 and λ_2 be the arrival rate for sessions of type T_1 and T_2. Then, $P[T_1, T_2, x]$, the expected number of events where an arrival of type T_1 is followed by an arrival of type T_2 within time x, is $P[T_1, T_2, x] \leqslant \lambda_1 \lambda_2 x$.*

The proof can be found in Appendix A. Note that we also verified Theorem 1 using Monte Carlo simulations [16]. Next, we consider the scenario where there are $m \geqslant 2$ types of connections.

COROLLARY 1. *Let there be m types of sessions and let α be the threshold for reporting a connection pair. Then the rate of false positives per unit time can be upper bounded as $m^2\alpha$.*

To prove this corollary, first note that the number of different pairs of session types under consideration is m^2 (note that a session of a particular type can also be a "trigger" for a second session of the same type). Thus, from Theorem 1, since α is an upper bound on the rate of false positives for a particular ordered pair of session types, a simple union bound gives the formula in Corollary 1.

In practice, we choose the unit of time for measuring α as an hour and we found that $\alpha = 0.1$ per hour works well. Although our proofs of these theorems assume stationarity, our results apply even otherwise (e.g., non-homogeneous Poisson processes), as long as the rate estimation algorithm adapts to the changing rates. Finally, in our experiments, we found that the number of false positives is typically lower than the worst-case bound proved above.

4.4 Discussion

The fact that our Session Extraction approach is statistical imposes certain constraints on our abstraction approach. That it may exhibit false negatives is not particularly disconcerting: since our focus is on discovering application behavior, usually a trace will contain several instances of a particular type of behavior, so we can abide missing some. Our extraction need not find *all* instances to be successful. Further, our Session Extraction approach may also have false positives: for instance, the Poisson assumption used in our statistical test may not hold, or, it may so happen that two connections occur close-by in time simply by coincidence. We would not want our abstraction mechanism led astray into deducing session descriptors that incorrectly incorporate elements induced by false aggregations.

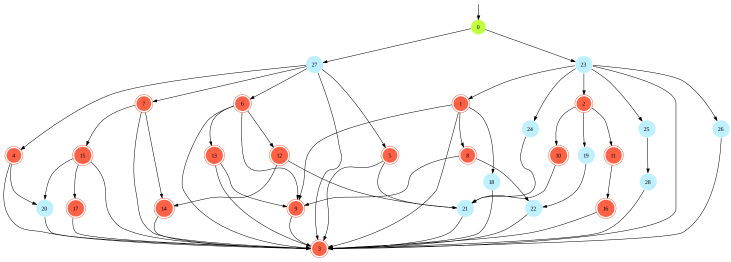

Figure 1: The Exact DFA for FTP

Our strategy is to choose α to obtain acceptable false negative performance, and to then design our abstraction mechanism to explicitly accommodate the level of false positives this leads to (for which the analysis in the previous section also gives a useful bound). We assessed different candidate values of α by manually assessing the false negatives for 4 applications (SMTP, FTP, two Web services) present in our traces, which led us to select $\alpha = 0.1$. This setting incurs a false negative rate of less than 25% [16].

5. STRUCTURE ABSTRACTION

The Structure Abstraction process aims to derive succinct descriptions for application sessions based on the set of session types reported by Session Extraction. We discuss how we represent session descriptions and then present our abstraction framework.

5.1 Representation of Session Descriptors

We first need to resolve "representation": what language should we use to abstractly describe the structure of sessions? We looked for a good balance between expressiveness and ease of generating abstractions from complex initial descriptions, which led us to choose regular expressions.[1] This choice was supported by our previous empirical experiences when we manually attempted to derive descriptions for different types of sessions [16]. In our discussions and figures we will often use the DFA equivalents of particular regular expressions. We also further refine this representation by labeling state transitions with probabilities, similar to the Customer Model Behavior Graphs used for characterizing workloads on websites [20].

Thus, given a set **ST** of observed session types **ST** $= (ST_1, ST_2, \dots)$, we can capture the full structural range using a regular expression that explicitly matches the entire set. Figure 1 shows such as complete DFA for FTP (as derived from our larger dataset). Here we have omitted the labeling because the point of the figure is simply to convey the great complexity that the full structural range can manifest. In this case, the DFA is complex (with 28 states) due to the fact that it has to *exactly* capture several FTP sessions varying in the number and the direction of data transfers.

Figure 2, on the other hand, shows a more "natural" (and tractable) DFA that abstracts much of the original while preserving some of its unintuitive features (the presence of HTTP transitions,

[1] We also experimented with using Hidden Markov Model inference techniques to abstract sessions, but found the results much harder to intuitively understand, as well as requiring computation that scales poorly with the size of the trace.

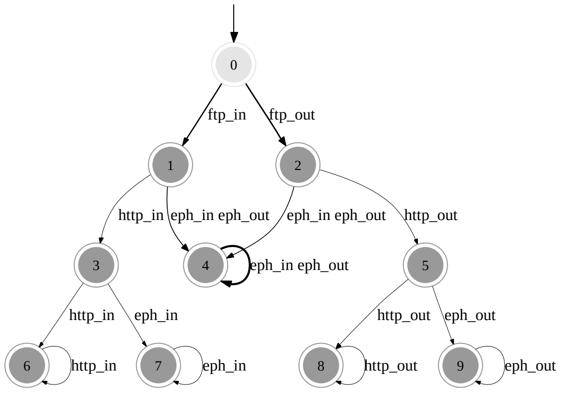

Figure 2: Abstract Session Descriptor for FTP

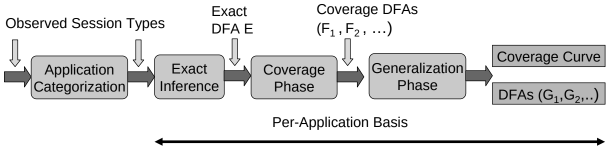

Figure 3: Overview of Structure Abstraction Framework

for example). In the diagram, text labels are associated with the transitions to their left. A label like $http\_in$ indicates an inbound HTTP connection. eph corresponds to an ephemeral port (one that is both not well-known and changes from session to session). Also, recall that the thickness of an arc indicates its relative frequency of transition in comparison to all the other arcs in the DFA.

The goal of our structure abstraction framework is to derive such a "natural" DFA since it yields a number of benefits over the exact DFA:

Simplicity: The reduction in number of states and edges makes the DFA easier to comprehend.

Generalization: The abstracted DFA can capture a more complete set of possible structures, including some not present in **ST**. For example, in Figure 2 the construct $(ftp\_in|ftp\_out)$ $(other\_in|other\_out)*$ captures an infinite set of session types; the "*" construct cannot occur in the exact DFA since it is derived from a finite trace.

Highlighting Common Behavior: Some types of sessions can exhibit quite different modes (e.g., browsing sessions), and/or instances of individualized or idiosyncratic behavior (e.g., login sessions that spawn many subsequent connections). By constructing abstractions that weight commonly seen elements over rare ones, we can highlight for the analyst tradeoffs between simplicity and capturing rare activity.

Minimizing False Positives: Given the statistical nature of our session extraction, the exact DFA may include false aggregations. Abstraction can help weed these out because they will tend to appear as isolated, rare structures, similar to the last item above.

5.2 Abstraction Framework

Figure 3 shows the four steps in our abstraction framework:

Application Categorization. This semi-automatic step identifies the applications in the trace and uses these to categorize the observed session types. The remaining steps operate on a per-application basis.

We lack ground truth for identifying the applications present in our trace at the granularity we desire, namely notions like "user is processing their email." Instead, we use the service associated with the first connection in a session as a proxy for the application type. To do so, we extracted a list of service ports by identifying those occurring in the trace more often than a fixed threshold, $\mathsf{T}_{service}$ (which we set to 5). We then manually analyzed this list to determine the associated application via either entries in an extensive directory [30], or, in ambiguous cases, by inspecting packet payloads when available. Armed with the list of types of applications, we then categorize sessions based on the server port of the first connection in that session.

Exact Abstraction. This step produces an exact DFA, E, that describes the session structure of an application A based on the complete set of session types, **ST** observed for A. We construct E from the union of each of the observed session types, minimizing the DFA using the FSA toolbox [12].

Coverage Phase. This step (detailed in §5.3) emits a sequence of DFAs $F_1, F_2, \ldots$ that represent subsets of E that progressively (and greedily) account for greater and greater *coverage* (fraction of the set **ST** matched) as we add edges.

Generalization Phase. This step (detailed in §5.4) applies three generalization rules to the sequence $F_1, F_2, \ldots$ that introduce commonly useful abstractions, producing a set of generalized DFAs, $G_1, G_2, \ldots$.

At the end of this process, we present to the analyst a *coverage curve* that plots the coverage of generalized DFAs against their complexity. The analyst then uses this curve as guidance regarding which DFAs to inspect in order to understand the application's session structure at different levels of abstraction. We discuss this process in §5.5.

5.3 Coverage Phase

Our coverage phase aims to extract a set of DFAs that capture subsets of the observed session behavior that best trade off simplicity-of-expression (fewest states/edges) for coverage (capturing most types of observed behavior). As noted above, this helps both with keeping rare or peculiar session instances from obscuring the commonly observed patterns, and for minimizing the effects of false positives in our extraction algorithm (since these will tend to manifest as rare, peculiar sessions). This phase produces a sequence of DFAs $F_1, F_2, \ldots, F_n$ that "cover" the observed set **ST** to increasing extents. We formulate the notion of a DFA F_i's "coverage metric" with respect to a set **ST** as the fraction of session types in **ST** accepted by F_i, weighted by the frequency with which the type occurs. Given this definition, the following greedy algorithm produces a sequence of DFAs with increasing coverage metric.

We first feed every session instance in **ST** to E, the exact DFA, accumulating a *hit count* $h(e)$ for every edge $e \in E$, i.e., the number of traversals of e. Next, for each edge e we compute the *augmented hit count*, $h'(e)$, as $\sum_{\text{e' reachable from e}} h(e')$. The purpose of this augmented hit count is to capture the implicit dependencies among the edges in the DFA. The overall idea is to prioritize "upstream" edges more than "downstream" edges. We then order edges by decreasing $h'(e)$; denote this ordering by $e_1, e_2, \ldots$. Finally, we construct DFAs F_i by taking the union of all edges $e_1, \ldots, e_i$.

5.4 Generalization Phase

In this step we subject the coverage DFAs, $F_1, F_2, \ldots$, to a set of transformations to obtain a sequence of generalized DFAs, $G_1, G_2, \ldots$. We found 3 generalizations that worked well across about 40 applications in our trace:

Prefix Rule. If we observe a session type $ST_i = (T^i_1, \ldots, T^i_m)$ in **ST**, then consider any prefix of this type also a session type for the application. We implement this rule by marking all states of the DFA F_i as accepting states.

Invert Direction Rule. We base this rule on the observation that an application session is typically independent of the direction (inbound vs. outbound) of the originating connection. For example, if we observe $\overline{ST_i} = (ftp\_in, other\_in)$, then we extend the DFA to also match $\overline{ST_i} = (ftp\_out, other\_out)$.

Counting Rule. If a DFA F_i matches aBc and aB^nc, (for $n > 1$), where a and c are individual connection types and B is a union of one-or-more connection types (e.g., $other\_in|other\_out$), then we transform F_i so that the corresponding G_i matches aB^+c. Note that in §6 we find that restricting $n = 2$ provides satisfactory results.

Finally, while the order in which we apply these rules affects the generalized DFA G_i, it can be shown that the following sequence is idempotent: Prefix Rule, Invert Direction, Counting Rule. Thus, we simply apply the rules in this sequence once and output the result as G_i. Note that these rules have an appealing monotonicity property: if the sequence of DFAs $F_1, F_2, \ldots$ has increasing coverage metrics, the sequence $G_1, G_2, \ldots$ retains this property.

In our experiments, we chose to apply all of these rules before presenting the DFA to the analyst. (An alternative would be to leave the choice of which rules to apply to the analyst.) The Prefix Rule intuitively holds since any session may terminate mid-way due to various error conditions. In our trace, we found only one case where the Prefix Rule not hold: one mail server always originated exactly two reverse Ident connections in response to an incoming SMTP connection. The Invert Direction rule is sometimes incorrect in the sense that a particular session structure visible in sessions originated from inside (outside) may not apply for sessions originated in the other direction. We however chose to apply it since the weights included on the edges do reflect this fact to the analyst. Also, the session structure for most applications in our trace also conform to the Counting Rule (the mail server we alluded to before does not conform to this rule).

5.5 Coverage Curves

A coverage curve, such as that shown in Figure 4(A), plots the number of edges i in the generalized DFA G_i against G_i's coverage metric. Note that though we obtain F_i (the basis for G_i) by retaining exactly i edges E, G_i may have more or fewer than i edges. This is because our generalization rules can simplify the structure of the DFA by adding edges (Invert Direction and Counting Rules) or marking certain states as final states (Prefix Rule). Since our DFAs are always minimized at every step in the process, performing the minimization procedure, after such a simplification of structure, may *decrease* the number of edges. This phenomenon is what leads to the non-monotone nature of the curve, as illustrated in the dip at 8 edges in Figure 4(A).

A knee in the coverage curve marks a point where the coverage metric increases sharply with the addition of a few particular edges. Such knees generally correspond to *modes*: points where adding a bit more complexity to the abstraction provides a substantially more comprehensive description. Such knees guide the analyst in choosing which DFAs merit scrutiny. In addition, the coverage curve helps the analyst deal with the problems of idiosyncratic sessions and false aggregations. Both of these typically appear towards the far right side of the coverage curve.

6. RESULTS

In this section, we describe the evaluation of our scheme using 4 weeks of traces collected at the border of the Lawrence Berkeley National Laboratory, a site with about 8,000 hosts that on average participated in 2,700,000 connections each day. We used the first 2 weeks for calibration and guiding the design of heuristics in our scheme, while we present results obtained by using our calibrated scheme over the last 2 weeks.

6.1 Parameter Settings

Session Extraction. For setting the timing parameters T_{aggreg}, $\mathsf{T}_{trigger}$, and T_{rate}, we were guided by [24, 29]. We verified that for the corresponding values, for most applications the arrival process of sessions was Poisson-like (see [16] for detailed results). A few applications, such as *ntp*, violated Poisson grossly because they are timer-based, but in general, the imprecision of our statistical session extraction test is remedied by the way in which our structure abstraction mechanism trims rarely exhibited behavior.

We set T_{aggreg} (used in the aggregation rule) to 100 sec. We also experimented with a few other values (200 and 500 sec) suggested in [24, 29], but in the range we experimented with, we found that our final session descriptors did not vary much.

We set $\mathsf{T}_{trigger}$ (the maximum duration between the finish time of a session and the arrival time of a new connection) to 500 sec. In general, for most application sessions, $\mathsf{T}_{trigger}$ need be no greater than 100 sec. However, a few applications like FTP and Login sessions, sometimes had long sessions in our trace. So, we conservatively set $\mathsf{T}_{trigger}$ to be 500 sec based on duration of sessions we observed in our trace. Note that the number of false positives in our mechanism is upper-bounded by the parameter α (irrespective of the value of $T_{trigger}$). Thus, despite conservatively setting $T_{trigger}$ to 500 sec, the false positive performance is still under the threshold α.

We set T_{rate} (the time duration over which rate estimates are computed) to 3,600 sec, the value over which the arrival rates were reported to be stationary in [24, 29].

We set the threshold used in our statistical test, α, to 0.1, based on our calibration of sessions for four applications. We first extracted, using our partial *a priori* knowledge of the session structure for SMTP, FTP, and two Web services (a Web proxy service, and an HTTP interface to a service running on 9303/tcp), all sessions belonging to these applications in the trace. We then examined how many of the *session types* implied by these legitimate sessions were found by our session extraction test. Except for FTP, it found all of them. For FTP, extraction generally missed long sessions consisting of several data-transfer connections, but such sessions have a very simple structure of the form (**ftp**) (**ftp-data**)*. Missing these is not a serious concern when inferring session descriptors, since our structure abstraction using its generalization rules can “fill in the gaps”.

Structure Abstraction. The parameters used in structure abstraction are relatively easier to set. Regarding the semi-automatic application categorization step, since our packet captures did not include the entire packet, we had to rely on port numbers to classify connections into applications. Once this classification was done, we looked at all applications occurring at least 5 times in the 2-week trace (i.e., $\mathsf{T}_{service} \geq 5$).

Apart from this parameter, there are two details worth noting. In our naïve implementation, the counting rule (inferring general positive closure) is too expensive to implement for high values of $|B|$ (the size of the repeating unit). Since the number of states s sometimes exceeds 100, we examined the impact of using much smaller values. We found that simply restricting the rule to the case of $|B| = 2$ allows us to correctly infer positive closure in each instance where we know *a priori* that it makes sense. In addition, we only feed our structure abstraction algorithm session types of length ≤ 10—otherwise, the number of states in the DFA can explode due to the exponential growth in the number of session types with increasing length. One example of such a long session is an FTP session where a control connection is followed by, say, 15 data connections. We find that omitting such sessions does not result in any loss of information in the inferred session descriptors, since our generalization rules can usually capture such session structures anyway (in this case, by allowing any number of data connections).

Role of Analyst. In obtaining the results below, the role of the analyst is to first use the coverage curve in order to select a cutoff coverage fraction, and then peruse the appropriate DFAs corresponding to that fraction. In some cases, the existence of knees in the coverage curve makes this choice easy to make, while in others (such as HTTP), the presence of a long tail implies that the analyst can choose as much detail to "explore" as much of the tail as he desires.

6.2 Empirically Observed Session Structure

We now turn to examining some of the session structures discovered for the applications in our trace. For session structures that can be verified by using protocol specifications, we found that our session descriptions generally capture *all* the behavior implied by the specification with no false positives. Unfortunately, many session structures arise due to empirical behavior, and for those, we can only assess their plausibility. Finally, some sessions arise for truly anomalous reasons, such as misconfigurations or attacks. These last will typically be few in number, and will not appear in the DFA unless the analyst asks for very high coverage. However, they are interesting in demonstrating the value of having a general tool that can detect causality.

6.2.1 FTP

Figure 4 shows the coverage curve (subplot A) for FTP , and DFAs representing some knees in this graph in subplots B-G. We use this as our primary example of the possibilities of our session structure discovery.

The coverage curve shows the number of edges in the generalized DFA G_i versus the coverage provided by G_i. We see a linear increase until 6 edges, after which the coverage tapers off slowly until about 100 edges (tail not shown in the figure). We now examine the various knees in this curve, showing that they usually correspond to some feature of the underlying sessions. Note that we describe the generalized DFAs corresponding to the knees in this graph as ordered by the number of edges included from the exact DFA; this does not necessarily correspond to ordering them by the number of edges in the DFA itself, since the generalization procedure can sometimes simplify the DFA considerably.

The first noteworthy point occurs at 2 edges. This is simply the DFA (not shown) that captures singleton incoming and outgoing FTP sessions. The second point occurs at 4 edges (subplot B), corresponding to the DFA capturing sessions with a single data transfer connection in the same direction as the initial control connection. Subplot C shows the next DFA of interest, which also has 4 edges (the plot shows the highest coverage exhibited by a DFA of a given number of edges; so the coverage in the plot for 4 edges corresponds to this DFA). This captures the pattern $(ftp\_in|ftp\_out)(eph\_in|eph\_out)$, which allows for sessions with a single data transfer in either direction. Subplot D shows the next interesting DFA, with 8 edges, which also captures incoming (outgoing) FTP sessions with a single data transfer in the opposite direction. The DFA in Subplot E includes more edges from the exact DFA, but has fewer actual edges due to generalization of the structure of the DFA, capturing sessions with any number of data transfers in the same direction. The DFA in Subplot F (10 edges) shows how HTTP connections can occur during FTP sessions, likely due to intermingled access to Web pages with links to FTP URLs. Finally, another knee (not particularly visible in the coverage curve) occurs at 18 edges, as shown in subplot G. This DFA captures sessions with any number of FTP or HTTP transfers in either direction. This DFA has over 99% coverage, implying that it "explains" nearly all the sessions found by session extraction, and, further, captures all the characteristics of the FTP protocol specification.

6.2.2 Timbuktu

Figure 5 shows the coverage curve and two pertinent DFAs for Timbuktu [36], a Mac and Windows remote desktop application. The coverage curve (subplot A) shows a sharp knee in the beginning, gradually tapering off towards the tail. The knee corresponds to singletons, which comprise > 90% of sessions. Although the tail accounts for < 10% of the remaining sessions, it reveals interesting details. Subplot B shows the DFA with 4 edges, revealing that Timbuktu sessions may include some browsing behavior as well. Subplot C shows the DFA with 10 edges. The associated ephemeral ports likely correspond to dynamic ports negotiated in the main channel (which corresponds to the Timbuktu listening port). We also see browsing behavior reflected by associated HTTP connections, and that Timbuktu connections can occur in conjunction with the AppleTalk file sharing protocol, presumably due to users performing file transfers along with remote control software. Finally, SSH connections also occur in such sessions, suggesting that login connections of several kinds tend to occur together.

6.2.3 HTTP

HTTP sessions come in a number of variations. By far, the most common ($\approx 99\%$) are singleton or aggregated sessions that reflect successive retrieval of multiple pages from the same server, which the coverage curve (omitted for space constraints) shows as a very sharp knee very early on. However, there is also a long tail clearly visible in the coverage curve, accounting for the other 1% of sessions. To illustrate the useful information that may be gleaned from this tail, Figure 6 presents the HTTP DFA with 30 edges, though given limited space we show only half the edges, those corresponding to sessions begun with an outgoing HTTP connection. We chose this DFA simply to illustrate the variety of HTTP sessions; it does not correspond to any obviously visible knee in the coverage curve. Although these sorts of DFAs in general reflect remote tail behavior, for some particular hosts they can be quite prominent (e.g., a server or a crawler). Highlighting such host-specific behavior for an analyst is a promising area for our future work.

The figure highlights that HTTP (port 80) can occur in conjunction with multiple connections on other ports (81, 8000, 8080, HTTPS, FTP) typically used for Web access. Likely the presence of ephemeral ports in the DFA also reflects this sort of linking, for example due to services offering HTTP interfaces that include client-side code (such as JavaScript) for retrieving additional data from the service. Connections to a port 8765 (marked "ultraseek") likely reflect an Ultraseek search engine crawler (supported by our observation that the hosts exhibiting this traffic have names that suggest indexers). We also see LDAP connections, perhaps reflecting Web services that rely on it for authentication. Finally, we see outgoing SSH connections. These may simply reflect a user's "start up" routine of opening both some Web pages and logging into a commonly

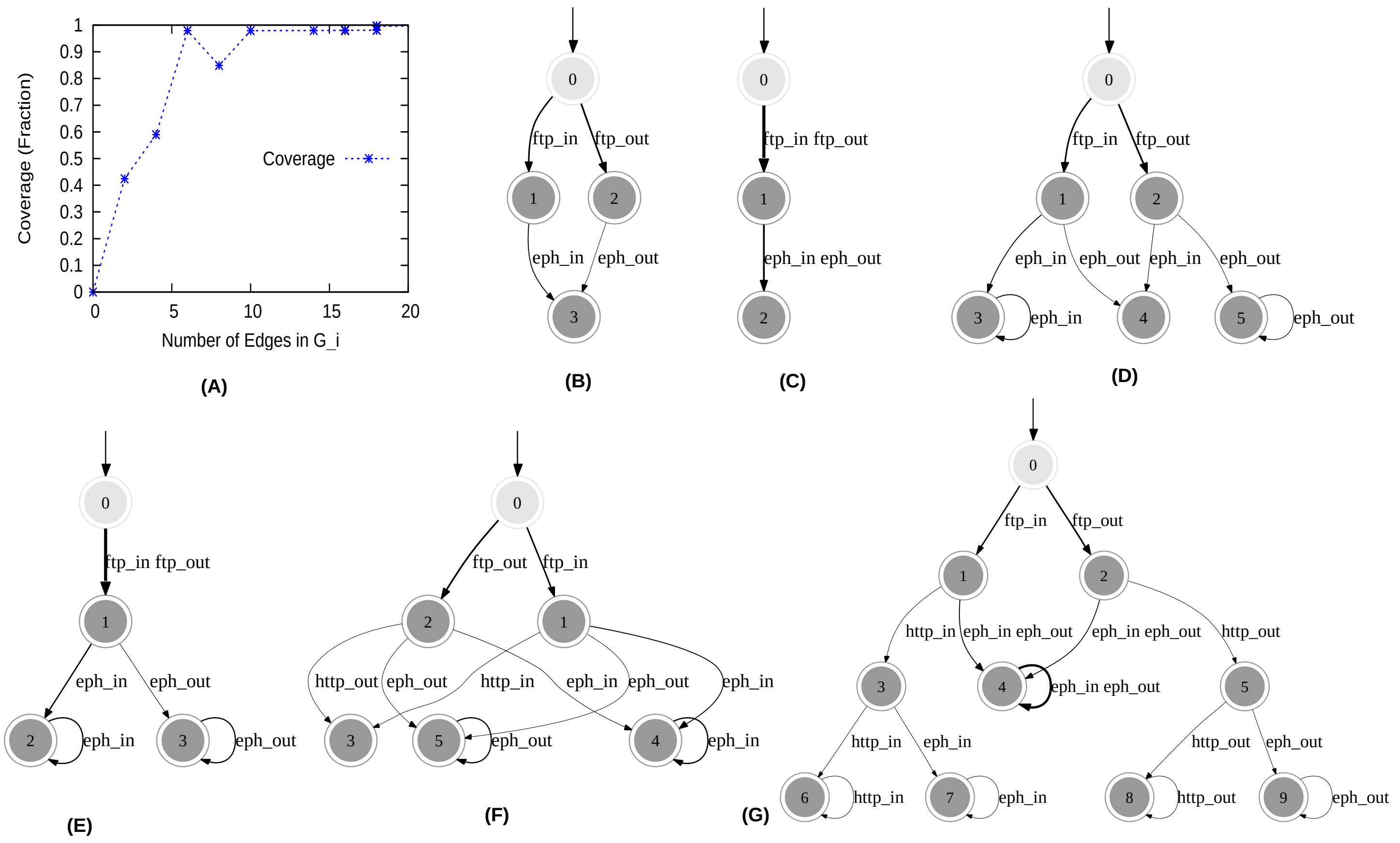

Figure 4: FTP Coverage Curve (A), DFAs: 4 edges (B), 4 edges (C), 6 edges (D), 8 edges (E), 10 edges (F), 18 edges (G)

visited remote host.

Deeper in the coverage curve, the DFAs also reflect AOL and other connections occurring in conjunction with HTTP, perhaps reflecting chatting and browsing alongside, and a greater variety of services offered via Web pages (e.g., Oracle database access and music streaming applications that use Real Time Streaming Protocol and Windows Media).

Other Applications. Due to space constraints, we summarize some of the more interesting additional application session structures that our scheme exposed in the trace, which included mail-related applications (SMTP, IMAP4, POP3), remote access (SSH, Exec, Rlogin, Telnet), database (ORACLE, MYSQL), bulk transfer (Grid FTP), Windows (NetMeeting, NETBIOS), and peer-to-peer (BitTorrent, KCEasy).

Mail-related session structures due to client-side applications (e.g., ThunderBird or Outlook) exhibit session structures of the form "*smtp_in (imap4_in | imap4ssl_in | pop3_in | pop3ssl_in)** " , reflecting the different protocol clients use to send and receive email. We also see server-side sessions of the form "*smtp_in (smtp_out | ident_out)*," reflecting SMTP mail relays and Ident reverse connections.

Remote access applications typically exhibit session structures such as "*ssh_in (ident_out | X11_out*)*," due to reverse Ident connections initiated by servers, and X11 connections initated by the users back to their desktop X11 servers. We also see structures such as "*ssh_in (ssh_out | vnc_in)*," which exhibit a "stepping-stone"-like structure [35, 42], though without the goal of "laundering" traffic since the users connect back to their own originating site. Note that, in the regular expression above, VNC stands for Virtual Network Computing, a popular remote desktop protocol.

Presumably such activity reflects creating multiple login windows or transferring files to supplement the login session. We also find structures such as "*ssh_in (ftp_in | http_in)*," presumably due to user browsing behavior once logged. Finally, GridFTP is dominated by sessions with a single outbound connection followed by multiple inbound ephemeral connections (though sometimes this all occurs in the opposite direction); services such as Oracle, SQL, and NetMeeting exhibit session structures that include multiple ephemeral connections alongside the primary connection on a well-known port; and P2P applications display the session structure of the form "*app_out app_in**."

6.3 Finding Attacks Using Anomaly Detection

Our experimental analysis over these traces also revealed sessions exhibiting anomalous structure reflecting malicious activity. Indeed, our original goal had been to detect network attacks by finding sessions that deviate from established session structures. Our hypothesis was that such deviations would reflect either unintended misconfigurations (a host behaving as spam relay or as a Web proxy), scanning, or "phone home" connections associated with compromises. Figure 7 shows an example of such an attack (confirmed by the site). The event consists of an incoming *ssh* connection, which compromises the host, followed by the host visiting a Web server (presumably controlled by the attacker), an incoming port 65535 connection (likely the attacker instructing the bot software they installed on the host) and then an outbound IRC con-

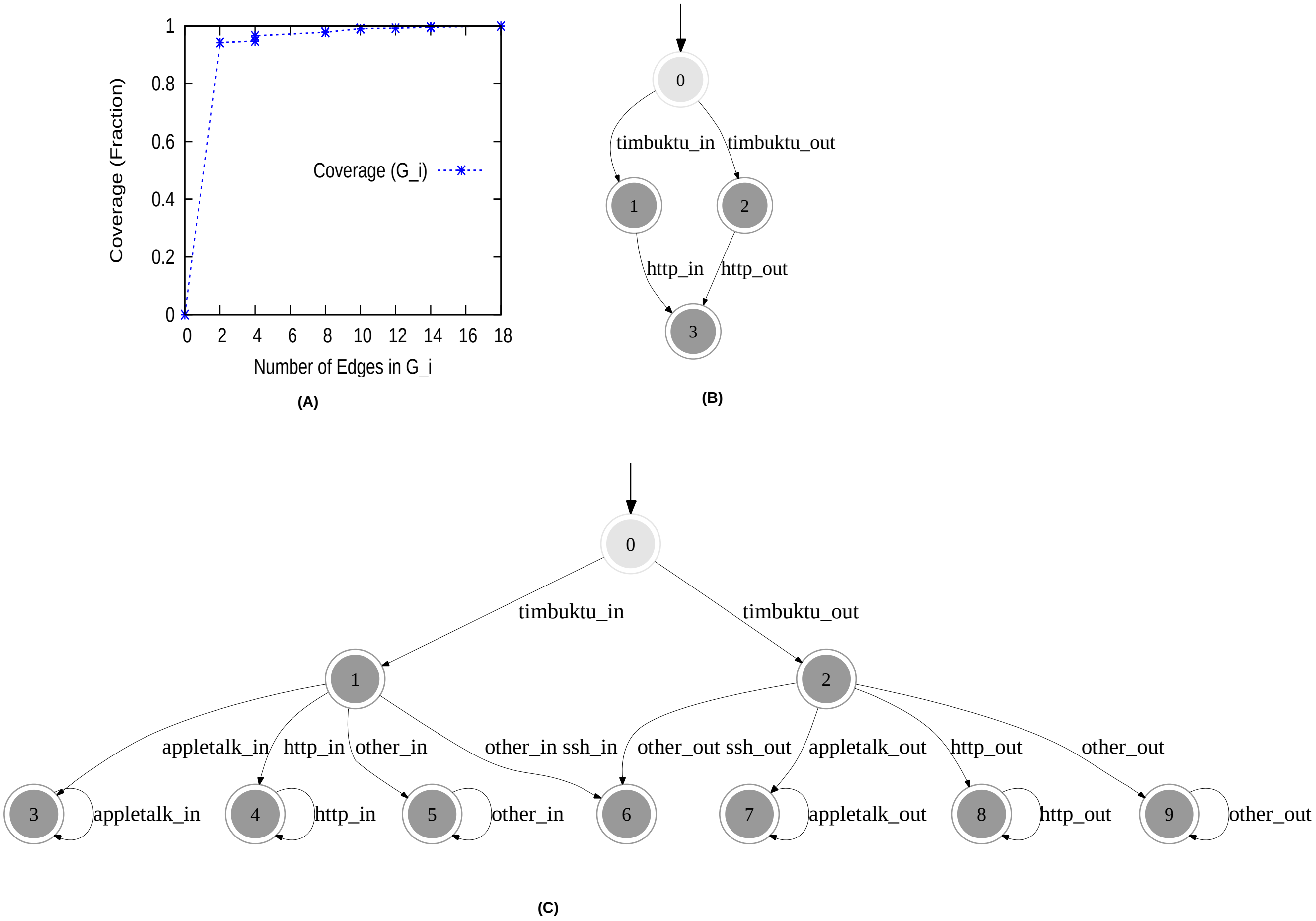

Figure 5: Timbuktu Coverage Curve (A) and DFAs: 4 edges (B), 18 edges (C)

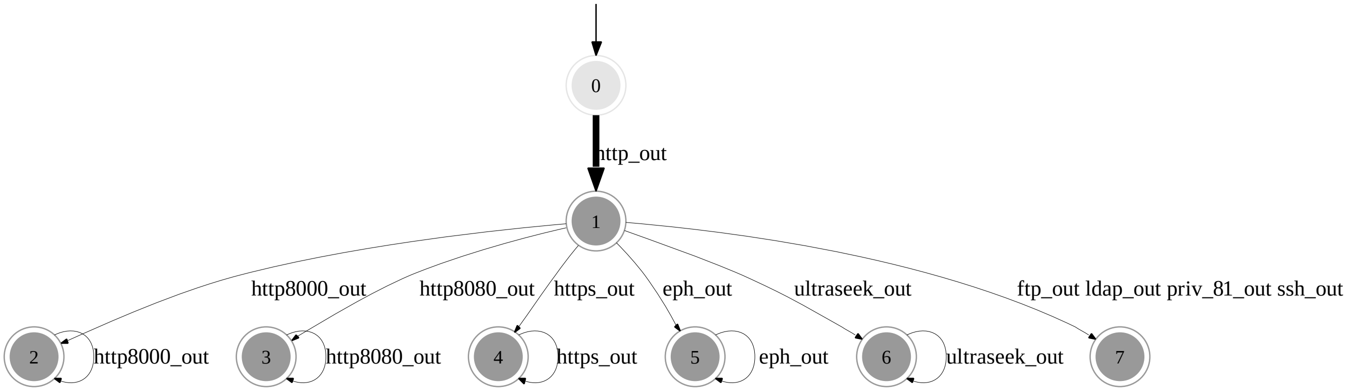

Figure 6: HTTP DFA with 30 edges

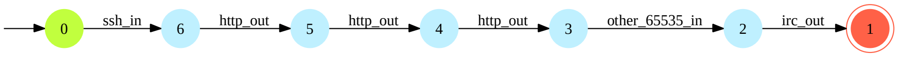

Figure 7: Session Illustrating Anomalous Behavior

nection (presumably to a botnet).

In assessing the sessions uncovered by our extraction algorithm, we have also found anamolous sessions caused by peer-to-peer applications disallowed by the site's policy (and using non-standard ports), and an instance where machine within the site acted as a HTTP relay for Yahoo Web pages, for reasons we could not determine.

Thus, although our session extractor, being statistical, might not catch all anomalous sessions, the sessions it does catch can be of considerable operational interest. Unfortunately, we find that such activity often only manifests in the upper tail of the coverage curve. Thus, there are usually too many strange-but-benign sessions—plus structures (perhaps) falsely inferred due to the statistical limits of our detection algorithm—for upper-tail instances to serve as directly "actionable" indications of attacker activity. We do hold hope, however, that detection of peculiar sessions can serve as input into further analysis that might use additional complementary information to derive actionable conclusions. Such information might be, for example, history about the host (e.g., what services it typically runs, and which remote hosts it has contacted in the past).

7. DISCUSSION

A major utility of session descriptors is in enabling the development of source models that can generate synthetic traffic that is similar to real traffic at the connection-level. Our work only captures the qualitative nature of such sessions; however, with such session descriptors in hand, one can then set about obtaining distributions for the duration of a session, number of connections in a session, and so forth (using techniques similar to those used in Nuzman *et al.* [24] for HTTP sessions). The main value of our approach is that it requires much less *a priori* information about the applications in the trace (it requires only a mapping from port number to application, whereas previous work explicitly required a session descriptor for the application as well).

The DFAs generated by our scheme include weights on the edges that can be used in order to generate "characteristic" synthetic sessions using the augmented hit counts we discussed in §5.3. This can be achieved by simply simulating a "probabilistic" DFA, where we determine the next state using weighted sampling from the various outgoing edges, terminating the procedure when none of the outgoing edges are chosen. This procedure only captures the qualitative structure; for self-loops that allow a variable number of connections, the distribution of the number of connections needs to be explicitly determined. We also note that since we infer our DFAs based on the output of session extraction—whose false negative performance may not be uniform across all types of sessions—this may bias the hit counts, depending on the degree to which the particular application conforms with our Poisson assumptions. In such cases, it may be possible to improve on the weights in these DFAs by comparing synthetic output with traces.

A particularly fruitful area for near-term future work is extending our methods to extract application sessions that involve multiple remote hosts (per the discussion in §4.3). Our work in [16] gives some preliminary results in this regard, finding that this extension better captures the behavior of peer-to-peer applications, proxies (mail and Web), and a number of other applications. For example, proxies typically demonstrate the characteristic session structure of an incoming connection from one remote host followed by an outgoing connection on the same port to another remote host.

More broadly, we hope that our session extraction and structure abstraction tools comprise a useful addition to the toolbox of administrators and researchers. Our session extraction approach can be thought of as inferring "hidden causality" in network connections; it identifies causally related connections by exploiting the observation that they typically occur closer to each other in practice compared to causally unrelated connections. This observation is the basis of our statistical test, which is by no means a bullet-proof causality test, in that it has non-negligible false negative ratio (and in the adversarial input case, an adversary simply needs to ensure that this connections are far apart enough in time to evade our test). However, its false positive ratio is bounded (as shown in §4.3), and thus most of the inferred sessions are indeed casually related connections. Furthermore, in cases such as worm propagation, where (if we extend the model to include multiple remote hosts) we can relate outgoing infection attempts to the initial infection connection, the scale of the number of connections originated by the host and the speed of the post-compromise activity may prove readily and quickly detectable by our session extraction mechanism.

8. CONCLUSION

In this work we have demonstrated a statistical technique to extract application sessions from a connection-level trace of network activity, and shown how to deduce descriptors that can be used by an analyst to capture the qualitative structure of such sessions. Our results demonstrate that a simple Poisson-based statistical technique, combined with a greedy approach for inferring DFA descriptors, works well over many of the applications in our trace. The results were, in some ways, surprising to us in terms of the various ways in which a particular application's structure manifests in our traces, and it is for this reason the validation of our results is still incomplete. Sessions reflecting protocol specifications are relatively easy to validate (although even this is hard for closed-source applications), while others reflecting user behavior (or) server-side configuration (or) client-side configuration are considerably more difficult to validate purely from in-network data.

In the future, we aim to evaluate and validate our methods over more applications; extend our approach to the case where a local host interacts during a session with multiple remote hosts; and devise mechanisms to collate descriptors for closely-related protocols. More broadly, the fact that our methods offer a general "causality" tool that can detect some forms of anomalous sessions also holds promise for detecting a variety of network attacks.

9. ACKNOWLEDGMENTS

We would like to thank Mark Allman, Rodrigo Fonseca, Dilip Joseph, Christian Kreibich, Karthik Lakshminarayanan, Ruoming Pang, Ananth Rao, Robin Sommer, Ion Stoica, and Mythili Vutukuru, for their helpful comments about the paper. We would also like to thank Manikandan Narayanan for helpful discussions. This work was supported in part by grants STI-0334088, ITR/ANI-0205519 and NSF-0433702 from the National Science Foundation, for which we are grateful.

10. REFERENCES

[1] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proc. SIGMETRICS*, pages 151–160, New York, NY, USA, 1998. ACM Press.

[2] A. Blum, D. Song, and S. Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. In *7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, pages 20–29, Sep 2004.

[3] J.-C. Bolot. End-to-end packet delay and loss behavior in the internet. In *Proc. SIGCOMM*, Sept 1993.

[4] K. Claffy, H.-W. Braun, and G. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE JSAC*, 13(8):1481–1494, 1995.

[5] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, C. Shannon, and J. Brown. Can we contain Internet worms? In *Proc. HOTNETS*, 2004.

[6] J. R. Crandall and F. T. Chong. Minos: Control Data Attack Prevention Orthogonal to Memory Model. In *Proc. International Symposium on Microarchitecture*, Oregon, Dec 2004.

[7] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proc. SIGMETRICS*, May 1996.

[8] W. Cui, V. Paxson, N. Weaver, and R. H. Katz. Protocol-Independent Adaptive Replay of Application Dialog. In *Proc. NDSS*, San Deigo, CA, Feb 2006.

[9] P. Danzig, S. Jamin, R. Caceres, D. Mitzel, and D. Estrin. ”an empirical workload model for driving wide-area tcp/ip network simulations”. *Internetworking: Research and Experience*, 3(1):1–26, 1992.

[10] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, 2001.

[11] H. Fowler and W. Leland. Local area network traffic characteristics, with implications for broadband network congestion management. *IEEE JSAC*, 9(7):1139–1149, 1991.

[12] Finite State Automata Utilities (version 6.2). `http://odur.let.rug.nl/~vannoord/Fsa/`.

[13] Graphviz - Graph Visualization Software. `http://www.graphviz.org/`.

[14] S. Heimlich. Traffic characterization of the nsfnet national backbone. In *Proc. Winter USENIX Conference*, 1990.

[15] R. Jain and S. Routhier. Packet trains —measurements and a new model for computer network traffic. *IEEE JSAC*, 4(6):986–995, 1986.

[16] J. Kannan, J. Jung, V. Paxson, and C. E. Koksal. Detecting hidden causality in network connections. Technical report, University of California, Berkeley, 2005.

[17] A. Kumar, V. Paxson, and N. Weaver. Exploiting underlying structure for detailed reconstruction of an internet-scale event. In *Proc. ACM IMC*, Oct 2005.

[18] W. Leland, M. Taqqu, W. Willinger, and D. Wilson”. ”on the self-similar nature of ethernet traffic”. *IEEE/ACM Transactions on Networking*, 2(1), 1994.

[19] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. Voelker. Automatic protocol inference: Unexpected means of protocol inference. In *Proc. ACM IMC*, Oct 2006.

[20] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. A. Mendes. A methodology for workload characterization of e-commerce sites. In *Proc. ACM conference on Electronic commerce*, pages 119–128, 1999.

[21] J. Mogul. End-to-end internet packet dynamics. In *Proc. SIGCOMM*, Sept 1997.

[22] A. W. Moore and D. Zuev. Internet traffic classification using Bayesian analysis techniques. In *SIGMETRICS ’05*, pages 50–60, 2005.

[23] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proc. NDSS*, 2005.

[24] C. Nuzman, I. Saniee, W. Sweldens, and A. Weiss. A compound model for TCP connection arrivals for LAN and WAN applications. *Computer Networks*, 40(3):319–337, 2002.

[25] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP throughput: A simple model and its empirical validation. In *Proc. ACM SIGCOMM*, 1998.

[26] V. Paxson. Empirically-derived analytic models of wide-area tcp connections. *IEEE/ACM Transactions on Networking*, 2(4), 1994.

[27] V. Paxson. Automated packet trace analysis of tcp implementations. In *Proc. SIGCOMM*, pages 167–179, New York, NY, USA, 1997. ACM Press.

[28] V. Paxson. End-to-end internet packet dynamics. In *Proc. SIGCOMM*, Sept 1997.

[29] V. Paxson and S. Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[30] TCP Ports List, UDP Ports List. `http://seifried.org/security/ports/`.

[31] The Protocol Informatics Project. `http://www.baselineresearch.net/PI/`.

[32] S. M. Ross. *Introduction to Probability Models, 8th Edition*. Academic Press, 2003.

[33] F. D. Smith, F. Hernandez-Campos, K. Jeffay, and D. Ott. What TCP/IP protocol headers can tell us about the web. In *SIGMETRICS/Performance*, pages 245–256, 2001.

[34] S. Staniford, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – A graph-based intrusion detection system for large networks. In *Proc. National Information Systems Security Conference*, 1996.

[35] S. Staniford-Chen and L. T. Heberlein. Holding intruders accountable on the internet. In *Proc. IEEE Symposium on Security and Privacy*, Washington, 1995.

[36] Timbuktu Pro Remote Control Software. `http://www.netopia.com/software/products/tb2/`.

[37] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level. In *Proc. SIGCOMM*, Sept 1995.

[38] Y. Xie, V. Sekar, D. Maltz, M. Reiter, and H. Zhang. Worm Origin Identification Using Random Moonwalks. In *Proc. IEEE Security and Privacy*, Oakland, CA, May 2005.

[39] K. Yoda and H. Etoh. Finding a connection chain for tracing intruders. In *Proc. ESORICS*, pages 191–205, London, UK, 2000. Springer-Verlag.

[40] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of internet flow rates. In *Proc. ACM SIGCOMM*, 2002.

[41] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the constancy of internet path properties. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, 2001.

[42] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proc. 9th USENIX Security Symposium*, Denver, CO, Aug 2000.

APPENDIX

A. DERIVATION OF THE FALSE POSITIVE RATE

In this section we prove Theorem 1 in two steps: First, we consider the case of a single Poisson process with rate λ, and compute the probability that it has two arrivals within x time units of each other in the time interval $[0, \tau]$. Then, we generalize this to the scenario where there are two independent Poisson processes with different rates.

LEMMA 1. *Let $\{X_i, i \geqslant 1\}$ be the interarrival times of the Poisson process $\{N(t), t \geqslant 0\}$ with an arrival rate λ. Then, for some $\tau > 0$*

$$\Pr[X_i < x \quad \textit{for some } i \leqslant N(\tau)] \leqslant \lambda^2 \tau x. \tag{1}$$

PROOF. Conditional on $N(\tau) = n$, for any given $i \leqslant n$ we can write

$$\Pr[X_i > x | N(\tau) = n] = \left(\frac{\tau - x}{\tau}\right)^n = \left(1 - \frac{x}{\tau}\right)^n.$$

This holds since the conditional distribution for arrival epochs follows the order statistics for a joint uniform distribution of n random variables [32]. Therefore

$$\Pr[X_i < x | N(\tau) = n] = 1 - \left(1 - \frac{x}{\tau}\right)^n \tag{2}$$

$$\leqslant 1 - \left(1 - n\frac{x}{\tau}\right) = \frac{nx}{\tau}, \tag{3}$$

where (3) follows since the higher order terms in the Binomial expansion satisfy

$$\binom{n}{2}\left(\frac{x}{\tau}\right)^2 - \binom{n}{3}\left(\frac{x}{\tau}\right)^3 + \cdots \geqslant 0.$$

Since the time for the first arrival is identically distributed with the i^{th} interarrival time,

$$\Pr[X_i < x | N(\tau) = N] \leq N\frac{x}{\tau} \text{ , for } 1 \leq i \leq N - 1$$

Next, let us look at the probability that there exists such an interarrival time within all n arrivals.

$$\begin{aligned}
&\Pr[X_i < x \text{ for some } i \leq N(\tau) | N(\tau) = n] \\
&= \Pr\left[\bigcup_{i \leqslant n} \{X_i < x | N(\tau) = n\}\right] \\
&\leqslant \sum_{i \leqslant n} \Pr\left[X_i < x | N(\tau) = n\right] \qquad (4) \\
&= (n-1)n\frac{x}{\tau}, \qquad (5)
\end{aligned}$$

where (4) follows from the union bound.

Finally, using (5) we find a bound for the *unconditional* probability that there exists an interarrival of size smaller than x in the entire window of τ seconds, where for some $i \leq N(\tau)$,

$$\Pr[X_i < x] = \frac{x}{\tau}(\mathrm{E}[N(\tau)])^2 = \lambda^2 \tau x \tag{6}$$

where (6) follows since the variance of a Poisson random variable is equal to its mean. Note that the bound we find becomes tight if the average interarrival time is large compared to x, i.e., $\lambda x \lll 1$. In this regime, the probability that two or more occurrences of such interarrivals is highly unlikely. Thus, the higher order terms in the binomial expansion are negligible and the union bound is tight.

We now generalize the previous lemma to the case when there are two types of sessions, T_1 and T_2, seen at a given local host.

LEMMA 2. *Let $\{N_1(t), t \geqslant 0\}$ and $\{N_2(t), t \geqslant 0\}$ be two independent Poisson processes with rates λ_1 and λ_2 respectively. Let $P[N_1(\tau), N_2(\tau), x]$ be the probability that an arrival in $N_1(t), t \leqslant \tau$ is followed by an arrival of $N_2(t), t \leqslant \tau$ within x time units. Then,*

$$P[N_1(\tau), N_2(\tau), x] \leqslant \lambda_1 \lambda_2 \tau x. \tag{7}$$

PROOF. Consider the combined process consisting of arrivals from both N_1 and N_2. Since this aggregate process is Poisson with rate $(\lambda_1 + \lambda_2)$, the probability that an interarrival of less than x occurs is upper bounded by $(\lambda_1 + \lambda_2)^2 \tau x$ by Lemma 1. Among such pairs we need to count those for which the first arrival belongs to N_1 and the second arrival belongs to N_2. From the independence of the two processes,

$$\begin{aligned}
P[N_1(\tau), N_2(\tau), x] &\leqslant (\lambda_1 + \lambda_2)^2 \tau x \cdot \frac{\lambda_1}{\lambda_1 + \lambda_2} \cdot \frac{\lambda_2}{\lambda_1 + \lambda_2} \\
&= \lambda_1 \lambda_2 \tau x. \qquad (8)
\end{aligned}$$

Note that Theorem 1 is a direct consequence of Lemma 2. We simply plug $\tau = 1$ since we deal with the rate of false positives, i.e., number of false positives per time unit. This is the basis for the statistical test in our session extraction approach.