Enriching Network Security Analysis with Time Travel

Gregor Maier
TU Berlin / DT Labs

Robin Sommer
ICSI / LBNL

Holger Dreger
Siemens AG
Corporate Technology

Anja Feldmann
TU Berlin / DT Labs

Vern Paxson
ICSI / UC Berkeley

Fabian Schneider
TU Berlin / DT Labs

ABSTRACT

In many situations it can be enormously helpful to archive the raw contents of a network traffic stream to disk, to enable later inspection of activity that becomes interesting only in retrospect. We present a *Time Machine (TM)* for network traffic that provides such a capability. The TM leverages the heavy-tailed nature of network flows to capture nearly all of the likely-interesting traffic while storing only a small fraction of the total volume. An initial proof-of-principle prototype established the forensic value of such an approach, contributing to the investigation of numerous attacks at a site with thousands of users. Based on these experiences, a rearchitected implementation of the system provides flexible, high-performance traffic stream capture, indexing and retrieval, including an interface between the TM and a real-time network intrusion detection system (NIDS). The NIDS controls the TM by dynamically adjusting recording parameters, instructing it to permanently store suspicious activity for offline forensics, and fetching traffic from the past for retrospective analysis. We present a detailed performance evaluation of both stand-alone and joint setups, and report on experiences with running the system live in high-volume environments.

Categories and Subject Descriptors:
C.2.3 [Computer-Communication Networks]: Network Operations – *Network monitoring*

General Terms:
Measurement, Performance, Security

Keywords:
Forensics, Packet Capture, Intrusion Detection

1. INTRODUCTION

When investigating security incidents or trouble-shooting performance problems, network packet traces—especially those with full payload content—can prove invaluable. Yet in many operational environments, wholesale recording and retention of entire data streams is infeasible. Even keeping small subsets for extended time periods has grown increasingly difficult due to ever-increasing traffic volumes. However, almost always only a very small subset of the traffic turns out to be relevant for later analysis. The key difficulty is how to decide *a priori* what data will be crucial when subsequently investigating an incident *retrospectively*.

For example, consider the Lawrence Berkeley National Laboratory (LBNL), a security-conscious research lab ($\approx$ 10,000 hosts, 10 Gbps Internet connectivity). The operational cybersecurity staff at LBNL has traditionally used bulk-recording with `tcpdump` to analyze security incidents retrospectively. However, due to the high volume of network traffic, the operators cannot record the full traffic volume, which averages 1.5 TB/day. Rather, the operators configure the tracing to omit 10 key services, including HTTP and FTP data transfers, as well as myriad high-volume hosts. Indeed, as of this writing the `tcpdump` filter contains 72 different constraints. Each of these omissions constitutes a blind spot when performing incident analysis, one very large one being the lack of records for any HTTP activity.

In this work we develop a system that uses dynamic packet filtering and buffering to enable effective bulk-recording of large traffic streams, coupled with interfaces that facilitate both manual (operator-driven) and automated (NIDS-driven) retrospective analysis. As this system allows us to conveniently "travel back in time," we term the capability it provides *Time Travel*, and the corresponding system a *Time Machine* (TM)[1]. The key insight is that due to the "heavy-tailed" nature of Internet traffic [17, 19], one can record most connections in their entirety, yet skip the bulk of the total volume, by only storing up to a (customizable) cutoff limit of bytes for each connection. We show that due to this property it is possible to buffer several days of raw high-volume traffic using commodity hardware and a few hundred GB of disk space, by employing a cutoff of 10–20 KB per connection—which enables retaining a *complete* record of the vast majority of connections.

Preliminary work of ours explored the feasibility of this approach and presented a prototype system that included a simple command-line interface for queries [15]. In this paper we build upon experiences derived from ongoing operational use at LBNL of that prototype, which led to a complete reimplementation of the system for much higher performance and support for a rich query-interface. This operational use has also proven the TM approach as an invaluable tool for network forensics: the security staff of LBNL now has access to a comprehensive view of the network's activity that has proven particularly helpful with tracking down the ever-increasing number of attacks carried out over HTTP.

At LBNL, the site's security team uses the original TM system on a daily basis to verify reports of illegitimate activity as reported by the local NIDS installation or received via communications from

[1]For what it's worth, we came up with this name well before its use by Apple for their backup system, and it appeared in our 2005 IMC short paper [15].

external sites. Depending on the type of activity under investigation, an analyst needs access to traffic from the past few hours or past few days. For example, the TM has enabled assessment of illegitimate downloads of sensitive information, web site defacements, and configuration holes exploited to spam local Wiki installations. The TM also proved crucial in illuminating a high-profile case of compromised user credentials [5] by providing evidence from the past that was otherwise unavailable.

Over the course of operating the original TM system within LBNL's production setup (and at experimental installations in two large university networks), several important limitations of the first prototype became apparent and led us to develop a new, much more efficient and feature-enhanced TM implementation that is currently running there in a prototype setup. First, while manual, analyst-driven queries to the TM for retrieving historic traffic are a crucial TM feature, the great majority of these queries are triggered by external events such as NIDS alerts. These alerts occur in significant volume, and in the original implementation each required the analyst to manually interact with the TM to extract the corresponding traffic prior to inspecting it to assess the significance of the event. This process becomes wearisome for the analyst, leading to a greater likelihood of overlooking serious incidents; the analyst chooses to focus on a small subset of alerts that *appear* to be the most relevant ones. In response to this problem, our current system offers a direct interface between the NIDS and the TM: once the NIDS reports an alert, it can ask the TM to *automatically* extract the relevant traffic, freeing the analyst of the need to translate the notification into a corresponding query.

In addition, we observed that the LBNL operators still perform their traditional bulk-recording in parallel to the TM setup,[2] as a means of enabling occasional access to more details associated with problematic connections. Our current system addresses this concern by making the TM's parameterization dynamically adaptable: for example, the NIDS can automatically instruct the redesigned TM to suspend the cutoff for hosts deemed to be malicious.

We also found that the operators often extract traffic from the TM for additional processing. For example, LBNL's analysts do this to assess the validity of NIDS notifications indicating that a connection may have leaked personally identifiable information (PII). Such an approach reflects a two-tiered strategy: first use cheap, preliminary heuristics to find a pool of possibly problematic connections, and then perform much more expensive analysis on just that pool. This becomes tenable since the volume is much smaller than that of the full traffic stream. Our current system supports such an approach by providing the means to redirect the relevant traffic *back to the NIDS*, so that the NIDS can further inspect it automatically. By coupling the two systems, we enable the NIDS to perform *retrospective* analysis.

Finally, analysis of our initial TM prototype in operation uncovered a key performance challenge in structuring such a system, namely the interactions of indexing and recording packets to disk while simultaneously handling random access queries for historic traffic. Unless we carefully structure the system's implementation to accommodate these interactions, the rigorous real-time requirements of high-volume packet capture can lead to packet drops even during small processing spikes.

Our contributions are: *(i)* the notion of efficient, high-volume bulk traffic recording by exploiting the heavy-tailed nature of network traffic, and *(ii)* the development of a system that both supports such capture and provides the capabilities required to use it effectively in operational practice, namely dynamic configuration, and

[2]One unfortunate side-effect of this parallel setup is a significantly reduced disk budget available to the TM.

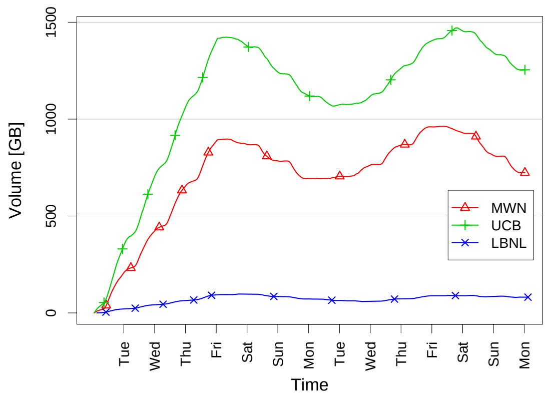


Figure 1: Required buffer size with $t_r = 4d$, 10 KB cutoff.

automated querying for retrospective analysis. We provide the latter in the context of interfacing the TM with the open-source "Bro" NIDS, and present and evaluate several scenarios for leveraging the new capability to improve the detection process.

The remainder of this paper is structured as follows. In §2 we introduce the basic filtering structure underlying the TM. We present a design overview of the TM, including its architecture and remote control capabilities, in §3. In §4 we evaluate the performance of the TM when deployed in high-volume network environments. In §5 we couple the TM with a NIDS. We discuss deployment trade-offs in §6 and related work in §7. We finish with a summary in §8.

2. EXPLOITING HEAVY-TAILS

The key strategy for efficiently recording the contents of a high-volume network traffic stream comes from exploiting the heavy-tailed nature of network traffic: most network connections are quite short, with a small number of large connections (the heavy tail) accounting for the bulk of total volume [17, 19]. Thus, by recording only the first N bytes of each connection (the *cutoff*), we can record most connections in their entirety, while still greatly reducing the volume of data we must retain. For large connections, we keep only the beginning; however, for many uses the beginning of such connections is the most interesting part (containing protocol handshakes, authentication dialogs, data items names, etc.). Faced with the choice of recording some connections completely versus recording the beginning of *all* connections, we generally prefer the latter. (We discuss the evasion risk this trade-off faces, as well as mitigation strategies, in §6.)

To directly manage the resources consumed by the TM, we configure the system with disk and memory *budgets*, which set upper bounds on the volume of data retained. The TM first stores packets in a memory buffer. When the budgeted buffer fills up, the TM migrates the oldest buffered packets to disk, where they reside until the TM's total disk consumption reaches its budgeted limit. After this point, the TM begins discarding the oldest stored packets in order to stay within the budget. Thus, in steady-state the TM will consume a fixed amount of memory and disk space, operating continually (months at a time) in this fashion, with always the most recent packets available, subject to the budget constraints.

As described above, the cutoff and memory/disk budgets apply to all connections equally. However, the TM also supports defining *storage classes*, each characterized by a BPF filter expression, and applying different sets of parameters to each of these. Such classes allow, for example, traffic associated with known-suspicious hosts to be captured with a larger cutoff and retained longer (by isolating its budgeted disk space from that consumed by other traffic).

We now turn to validating the effectiveness of the cutoff-based approach in reducing the amount of data we have to store. To assess this, we use a simulation driven off connection-level traces. The traces record the start time, duration, and volume of each TCP connection seen at a given site. Such traces capture the nature of their environment in terms of traffic volume, but with much less volume than would full packet-level data, which can be difficult to record for extended periods of time.

Since we have only connection-level information for the simulation, we approximate individual packet arrivals by modeling each connection as generating packets at a constant rate over its duration, such that the total number of (maximum-sized) packets sums to the volume transferred by the connection. Clearly, this is an oversimplification in terms of packet dynamics; but because we consider traffic at very large aggregation, and at time scales of hours/days, the inaccuracies it introduces are negligible [27].

For any given cutoff N, the simulation allows us to compute the volume of packet data currently stored. We can further refine the analysis by considering a specific *retention time* t_r, defining how long we store packet data. While the TM does not itself provide direct control over retention time, with our simulation we can compute the storage the system would require (i.e., what budget we would have to give it) to achieve a retention time of at least t_r.

For our assessment, we used a set of connection-level logs gathered between November 5–18, 2007, at three institutions: The *Münchner Wissenschaftsnetz (Munich Scientific Research Network, MWN)* connects two major universities and affiliated research institutes to the Internet (roughly 50,000 hosts). MWN has a 10 Gbps uplink, and its traffic totals 3–6 TB/day. Since our monitoring comes from a 1 Gbps SPAN port, data rates can reach this limit during peak hours, leading to truncation. The *Lawrence Berkeley National Laboratory* (LBNL) is a large research institute with about 10,000 hosts connected to the Internet by a 10 Gbps uplink. LBNL's traffic amounts to 1–2 TB/day. Our monitoring link here is a 10 Gbps tap into the upstream traffic. Finally, *UC Berkeley* (UCB) has about 45,000 hosts. It is connected to the Internet by two 1 Gbps links and has 3–5 TB of traffic per day. As SPAN ports of the two upstream routers are aggregated into one 1 Gbps monitoring link, we can again reach capacity limits during peak times.

The connections logs contain 3120M (UCB), 1898M (MWN), and 218M (LBNL) entries respectively. The logs reveal that indeed 91–94% of all connections at the three sites are shorter than a cutoff value of $N = 10$ KB. With a cutoff of 20 KB, we can record 94–96% of all connections in their entirety. (Of all connections, only 44–48% have any payload. Of those, a cutoff value of $N = 10$ KB truncates 14–19%; $N = 20$ KB truncates 9–13%.)

Fig. 1 plots the disk budget required for a target retention time $t_r = 4$ days, when employing a 10 KB cutoff. During the first 4 days we see a ramp-up phase, during which no data is evicted because the retention time t_r has not yet passed. After the ramp-up, the amount of buffer space required stabilizes, with variations stemming from diurnal patterns. For LBNL, a quite modest buffer of 100 GB suffices to retain 4 days of network packets. MWN and UCB have higher buffer requirements, but even in these high-volume environments buffer sizes of 1–1.5 TB suffice to provide days of historic network traffic, volumes within reach of commodity disk systems, and an order of magnitude less than required for the complete traffic stream.

3. THE TIME MACHINE DESIGN

In this section we give an overview of the design of the TM's internals, and its query and remote-control interface, which enables coupling the TM with a real-time NIDS (§5). What we present reflects a complete reworking of the original approach framed in [15], which, with experience, we found significantly lacking in both necessary performance and operational flexibility.

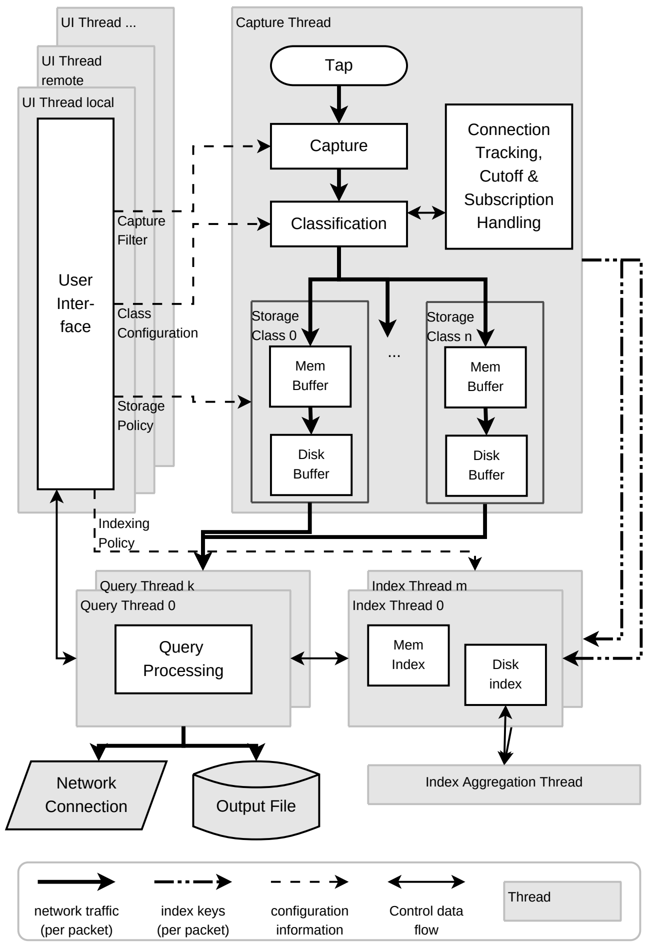


Figure 2: Architecture of the Time Machine.

3.1 Architecture

While in some ways the TM can be viewed as a database, it differs from conventional databases in that *(i)* data continually streams both into the system and *out* of it (expiration), *(ii)* it suffices to support a limited query language rather than full SQL, and *(iii)* it needs to observe real-time constraints in order to avoid failing to adequately process the incoming stream.

Consequently, we base the TM on the multi-threaded architecture shown in Fig. 2. This structure can leverage multiple CPU cores to separate recording and indexing operations as well as external control interactions. The *Capture Thread* is responsible for: capturing packets off of the network tap; classifying packets; monitoring the cutoff; and assigning packets to the appropriate storage class. *Index Threads* maintain the index data to provide the *Query Threads* with the ability to efficiently locate and retrieve buffered packets, whether they reside in memory or on disk. The *Index Aggregation Thread* does additional bookkeeping on index files stored on disk (merging smaller index files into larger ones), and *User Interface Threads* handle interaction between the TM and users or remote applications like a NIDS.

Packet Capture: The *Capture Thread* uses `libpcap` to access the packets on the monitored link and potentially prefilter them. It passes the packets on to Classification.

```
# Query. Results are sent via network connection.
query feed nids-61367-0 tag t35654 index conn4
    "tcp 1.2.3.4:42 5.6.7.8:80" subscribe

# In-memory query. Results are stored in a file.
query to_file "x.pcap" index ip "1.2.3.4" mem_only
    start 1200253074 end 1200255474 subscribe

# Dynamic class assignment.
set_dyn_class 5.6.7.8 alarm
```

Figure 3: Example query and control commands.

Classification: The classification stage maps packets to *connections* by maintaining a table of all currently active flows, as identified by the usual 5-tuple. For each connection, the TM stores the number of bytes already seen. Leveraging these counters, the classification component enforces the cutoff by discarding all further packets once a connection has reached its limit. In addition to cutoff management, the classification assigns every connection to a *storage class*. A storage class defines which TM parameters (cutoff limit and budgets of in-memory and on-disk buffers) apply to the connection's data.

Storage Classes: Each storage class consists of two buffers organized as FIFOs. One buffer is located within the main memory; the other is located on disk. The TM fills the memory buffer first. Once it becomes full, the TM migrates the oldest packets to the disk buffer. Buffering packets in main memory first allows the TM *(i)* to better tolerate bandwidth peaks by absorbing them in memory before writing data to disk, and *(ii)* to rapidly access the most recent packets for short-term queries, as we demonstrate in §5.4.

Indexing: The TM builds indexes of buffered packets to facilitate quick access to them. However, rather than referencing individual packets, the TM indexes all *time intervals* in which the associated index key has been seen on the network. Indexes can be configured for any subset of a packet's header fields, depending on what kind of queries are required. For example, setting up an index for the 2-tuple of source and destination addresses allows efficient queries for all traffic between two hosts. Indexes are stored in either main memory or on disk, depending on whether the indexed data has already been migrated to disk.

3.2 Control and Query Interface

The TM provides three different types of interfaces that support both queries requesting retrieval of stored packets matching certain criteria, and control of the TM's operation by changing parameters like the cutoff limit. For interactive usage, it provides a command-line console into which an operator can directly type queries and commands. For interaction with other applications, the TM communicates via remote network connections, accepting statements in its language and returning query results. Finally, combining the two, we developed a stand-alone client-program that allows users to issue the most common kinds of queries (e.g, all traffic of a given host) by specifying them in higher-level terms.

Processing of queries proceeds as follows. Queries must relate to one of the indexes that the TM maintains. The system then looks up the query key in the appropriate index, retrieves the corresponding packet data, and delivers it to the querying application. Our system supports two delivery methods: writing requested packets to an *output file* and sending them via a *network connection* to the requester. In both cases, the TM returns the data in `libpcap` format. By default, queries span all data managed by the system, which can be quite time-consuming if the referenced packets reside on disk. The query interface thus also supports queries confined to either specific time intervals or memory-only (no disk search).

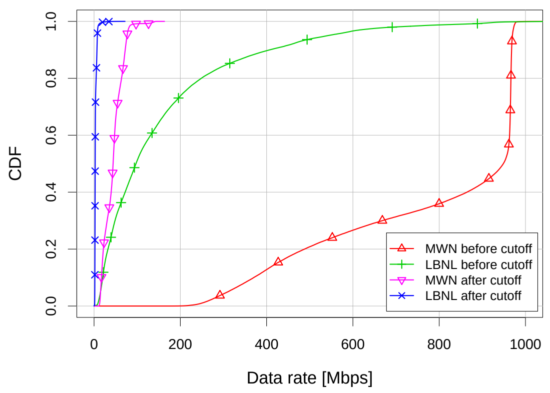


Figure 4: Bandwidth before/after applying a 15 KB cutoff.

In addition to supporting queries for already-captured packets, the query issuer can also express interest in receiving *future* packets matching the search criteria (for example because the query was issued in the middle of a connection for which the remainder of the connection has now become interesting too). To handle these situations, the TM supports query *subscriptions*, which are implemented at a per-connection granularity.

Queries and control commands are both specified in the syntax of the TM's interaction language; Fig. 3 shows several examples. The first query requests packets for the TCP connection between the specified endpoints, found using the connection four-tuple index `conn4`. The TM sends the packet stream to the receiving system `nids-61367-0` ("feed"), and includes with each packet the opaque tag `t35654` so that the recipient knows with which query to associate the packets. Finally, `subscribe` indicates that this query is a *subscription* for future packets relating to this connection, too.

The next example asks for all packets associated with the IP address `1.2.3.4` that reside in memory, instructing the TM to copy them to the local file `x.pcap`. The time interval is restricted via the `start` and `end` options. The final example changes the traffic class for any activity involving `5.6.7.8` to now be in the "`alarm`" class.

4. PERFORMANCE EVALUATION

We evaluate the performance of the TM in both controlled environments and live deployments at MWN and LBNL (see §2). The MWN deployment uses a 15 KB cutoff, a memory buffer size of 750 MB, a disk buffer size of 2.1 TB, and four different indexes (`conn4`, `conn3`, `conn2`, `ip`).[3] The TM runs on a dual-CPU AMD Opteron 244 (1.8 GHz) with 4 GB of RAM, running a 64-bit Gentoo Linux kernel (version 2.6.15.1) with a 1 Gbps Endace DAG network monitoring card [12] for traffic capture. At LBNL we use a 15 KB cutoff, 150 MB of memory, and 500 GB of disk storage, with three indexes (`conn4`, `conn3`, `ip`). The TM runs on a system with FreeBSD 6.2, two dual-core Intel Pentium D 3.7 GHz CPUs, a 3.5 TB RAID-storage system, and a Neterion 10 Gbps NIC.

4.1 Recording

We began operation at MWN at 7 PM local time, Jan. 11, 2008, and continued for 19 days. At LBNL the measurement started at Dec. 13, 2007 at 7 AM local time and ran for 26 days. While the setup at MWN ran stand-alone, the TM at LBNL is coupled with a NIDS that sends queries and controls the TM's operation as out-

[3] `conn4` uses the tuple (transport protocol, ip_1, ip_2, $port_1$, $port_2$); `conn3` drops one port; `conn2` uses just the IP address pair; and `ip` a single ip address. Note, each packet leads to *two* `conn3` keys and two `ip` keys.

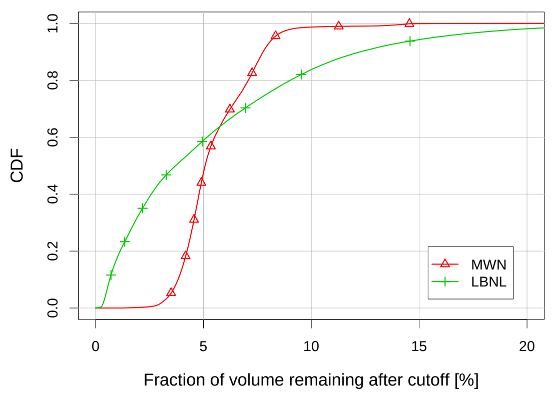


Figure 5: Traffic remaining after applying a 15 KB cutoff.

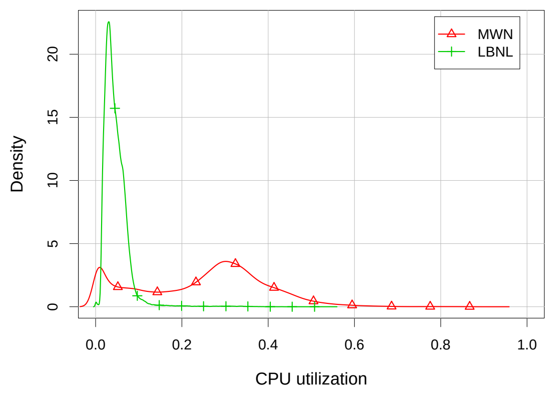


Figure 6: CPU utilization (across all cores).

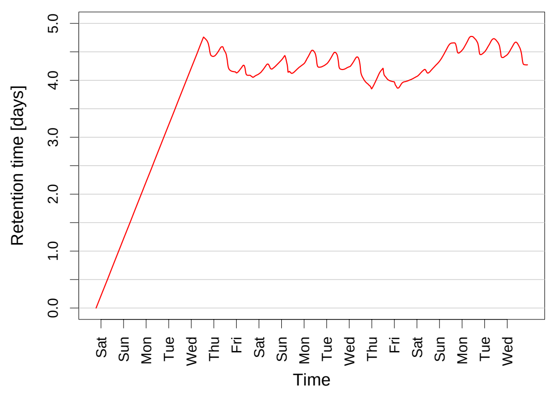


Figure 7: Retention time with 2.1 TB disk buffer at MWN.

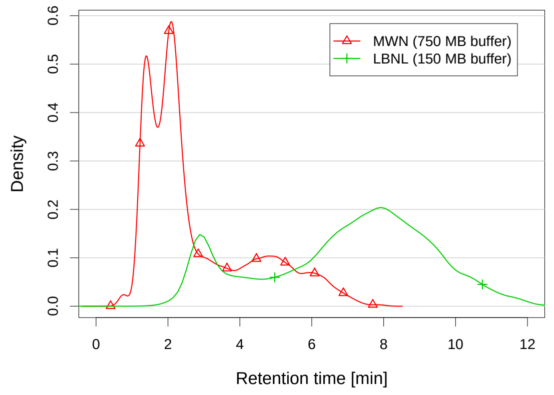


Figure 8: Retention in memory buffer.

lined in §5.1.[4] During the measurement period, the TM setup experienced only rare packet drops. At MWN the total packet loss was less than 0.04% and at LBNL less than 0.03%. Our investigation shows that during our measurement periods these drops are most likely caused by computation spikes and scheduling artifacts, and do not in fact correlate to bandwidth peaks or variations in connection arrival rates.

We start by examining whether the cutoff indeed reduces the data volume sufficiently, as our simulation predicted. Fig. 4 plots the original input data rates, averaged over 10 sec intervals, and the data rates after applying the cutoff for MWN and LBNL. (One can clearly see that at MWN the maximum is limited by the 1 Gbps monitoring link.) Fig. 5 shows the fraction of traffic, the reduction rate, that remains after applying the cutoff, again averaged over 10 sec intervals. While the original data rate reaches several hundred Mbps, after the cutoff less than 6% of the original traffic remains at both sites. Hereby, the reduction rate at LBNL exhibits a higher variability. The reduction ratio shows a diurnal variation: it decreases less during daytime than during nighttime. Most likely this is due to the prevalence of interactive traffic during the day which causes short connections while bulk-transfer traffic is more prevalent during the night due to backups and mirroring.

Next, we turn to the question whether the TM has sufficient resources to leave head-room for query processing. We observe that the CPU utilization (aggregated over all CPU cores, i.e., 100% reflects saturation of all cores) measured in 10 sec intervals, shown in Fig. 6, averages 25% (maximum $\approx$85%) for MWN indicating that there is enough head room for query processing even at peak times. For LBNL, the CPU utilization is even lower, with an average of 5% (maximum $\approx$50%). (The two local maxima for MWN in Fig. 6 are due to the diurnal effects.)

Fig. 7 shows how the *retention time* changes during the run at MWN. The 2.1 TB disk buffer provides $\approx$4 days during a normal work week, as one would expect given a $\approx$90% reduction in capture volume starting from 3–6 TB/day. After an initial ramp-up phase, the system retains an average of 4.3 days of network packets. As depicted in Fig. 8, the retention time in the *memory* buffer is significantly shorter: 169 sec of network traffic on average (41 sec minimum) for MWN. The local maxima are at 84 sec, and 126 sec respectively, due to the diurnal effects. At LBNL we achieve larger retention times. The 500 GB disk buffer retained a maximum of more than 15 days, and the 150 MB memory buffer (Fig. 8) was able to provide 421 sec on average (local maxima at 173 sec, and 475 sec).

Overall, our experience from these deployments is that the TM can satisfy queries for packets observed within the last days (weeks), providing that these are within the connection's cutoff. Moreover, the TM can answer queries for packets within the past couple of minutes very quickly as it stores these in memory.

4.2 Querying

As we plan to couple the TM with other applications, e.g., an intrusion detection system, that automatically generates queries it is important to understand how much load the TM can handle. Accordingly, we now examine the *query* performance of the TM with respect to *(i)* the number of queries it can handle, and *(ii)* the latency between issuing queries and receiving the corresponding replies. For these benchmarks, we ran the TM at LBNL on the same sys-

[4]During two time periods (one lasting 21 h, the other 4 days) the NIDS was not connected to the TM and therefore did not send any queries.

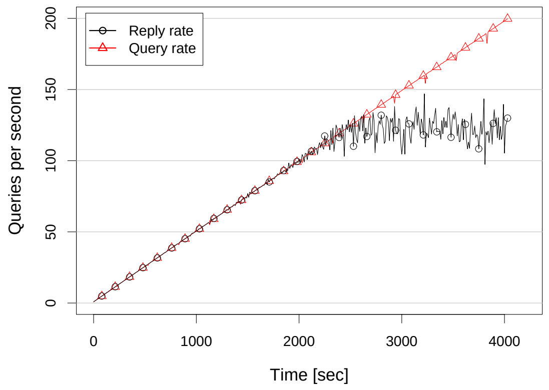


Figure 9: Queries at increasing rates.

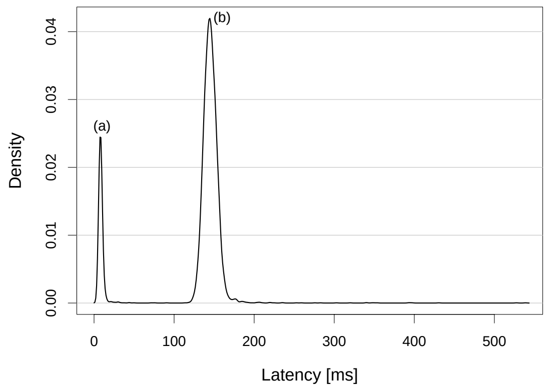


Figure 10: Latency between queries and replies.

tem as described above. For all experiments, we configured the TM with a memory buffer of 150 MB and a cutoff of 15 KB.

We focus our experiments on in-memory queries, since according to our experience these are the ones that are issued both at high rates and with the timeliness requirements for delivering the replies. In contrast, the execution of disk-based queries is heavily dominated by the I/O time it takes to scan the disk. They can take seconds to minutes to complete and therefore need to be limited to a very small number in any setup; we discuss this further in §6.

Load: We first examine the number of queries the TM can support. To this end, we measure the TM's ability to respond to queries that a simple benchmark client issues at increasing rates. All queries request connections for which the TM has data, so it can extract the appropriate packets and send them back in the same way as it would for an actual application.

To facilitate reproducible results, we add an *offline* mode to the TM: rather than reading live input, we preload the TM with a previously captured trace. In this mode, the TM processes the packets in the trace just as if it had seen them live, i.e., it builds up all of its internal data structures in the same manner. Once it finishes reading the trace, it only has to respond to the queries. Thus, its performance in this scenario may exceed its performance in a live setting during which it continues to capture data thus increasing its head-room for queries. (We verified that a TM operating on live traffic has head-room to sustain a reasonable query load in realistic settings, see §5.3.)

We use a 5.3 GB full trace captured at LBNL's uplink, spanning an interval of 3 min. After preloading the TM, the cutoff reduces the buffered traffic volume to 117 MB, which fits comfortably into the configured memory buffer. We configure the benchmark client to issue queries from a separate system at increasing rates: starting from one query every two seconds, the client increases the rate by 0.5 queries/sec every 10 seconds. To ensure that the client only issues requests for packets in the TM's memory buffer, we supplied it with a sample of 1% of the connections from the input trace. Each time the client requests a connection, it randomly picks one from this list to ensure that we are not unfairly benefiting from caching.

On the TM, we log the number of queries processed per second. As long as the TM can keep up, this matches the client's query rate. Fig. 9 plots the outcome of the experiment. Triangles show the rate at which queries were issued, and circles reflect the rate at which the TM responded, including sending the packets back to the client. We see that the TM can sustain about 120 queries/secs. Above that point, it fails to keep up. Overall, we find that the TM can handle a high query rate. Moreover, according to our experience the TM's performance suffices to cope with the number of automated queries generated by applications such as those discussed in §5.

Latency: Our next experiment examines query latency, i.e., the time between when a client issues a query and its reception of the first packet of the TM's reply. Naturally, we wish to keep the latency low, both to provide timely responses and to ensure accessibility of the data (i.e., to avoid that the TM has expunged the data from its in-memory buffer).

To assess query latency in a realistic setting, we use the following measurement with *live* LBNL traffic. We configure a benchmark client (the Bro NIDS) on a separate system to request packets from one of every n fully-established TCP connections. For each query, we log when the client sends it and when it receives the first packet in response. We run this setup for about 100 minutes in the early afternoon of a work-day. During this period the TM processes 73 GB of network traffic of which 5.5 GB are buffered on disk at termination. The TM does not report any dropped packets. We choose $n = 100$, which results in an average of 1.3 connections being requested per second ($\sigma = 0.47$). Fig. 10 shows the probability density of the observed query latencies. The mean latency is 125 ms, with $\sigma = 51$ ms and a maximum of 539 ms (median 143 ms). Of the 7881 queries, 1205 are answered within less than 100 ms, leading to the notable peak "(a)" in Fig. 10. We speculate that these queries are most likely processed while the TM's capture thread is not performing any significant disk I/O (indeed, most of them occur during the initial ramp-up phase when the TM is still able to buffer the network data completely in memory). The second peak "(b)" would then indicate typical query latencies during times of disk I/O once the TM has reached a steady-state.

Overall, we conclude that the query interface is sufficiently responsive to support automatic Time Travel applications.

5. COUPLING TM WITH A NIDS

Network intrusion detection systems analyze network traffic in real-time to monitor for possible attacks. While the real-time nature of such analysis provides major benefits in terms of timely detection and response, it also induces a significant constraint: the NIDS must immediately decide when it sees a network packet whether it might constitute part of an attack.

This constraint can have major implications, in that while at the time a NIDS encounters a packet its content may appear benign, *future* activity can cast a different light upon it. For example, consider a host scanning the network. Once the NIDS has detected the scanning activity, it may want to look more closely at connections originating from that source—*including those that occurred in the past*. However, any connection that took place prior to the time of detection has now been lost; the NIDS cannot afford to remember the details of everything it has ever seen, on the off chance that at some future point it might wish to re-inspect the activity.

The TM, on the other hand, effectively provides a very large buffer that stores network traffic in its most detailed form, i.e., as packets. By *coupling* the two systems, we allow the NIDS to access this resource pool. The NIDS can then tell the TM about the traffic it deems interesting, and in turn the TM can provide the NIDS with historic traffic for further analysis.

Given the TM capabilities developed in the previous section, we now explore the operational gains achievable by closely coupling the TM with a NIDS. We structure the discussion in five parts: *(i)* our prototype deployment at LBNL; *(ii)* experiences with enabling the NIDS to control the operation of the TM; *(iii)* the additional advantages gained if the NIDS can retrieve historic data from the TM; *(iv)* the benefits of tightly coupling the two systems; and *(v)* how we implemented these different types of functionality.

5.1 Prototype Deployment

Fig. 11 shows the high-level structure of coupling the TM with a NIDS. Both systems tap into the monitored traffic stream (here, a site's border) and therefore see the same traffic. The NIDS drives communication between the two, controlling the operation of the TM and issuing queries for past traffic. The TM then sends data back to the NIDS for it to analyze.

We install such a dual setup in the LBNL environment, using the open-source *Bro* NIDS [18]. Bro has been a primary component of LBNL's network monitoring infrastructure for many years, so using Bro for our study as well allows us to closely match the operational configuration.

The TM uses the same setup as described in §4: 15 KB cutoff, 500 GB disk budget, running on a system with two dual-core Pentium Ds and 4 GB of main memory. We interface the TM to the site's experimental "Bro Cluster" [26], a set of commodity PCs jointly monitoring the institute's full border traffic in a configuration that shadows the operational monitoring (along with running some additional forms of analysis). The cluster consists of 12 nodes in total, each a 3.6 GHz dual-CPU Intel Pentium D with 2 GB RAM.

We conducted initial experiments with this setup over a number of months, and in Dec. 2007 ran it continuously through early Jan. 2008 (see §4.1). The experiences reported here reflect a subsequent two-week run in Jan. 2008. During this latter period, the systems processed 22.7 TB of network data, corresponding to an average bitrate of 155 Mbps. The TM's cutoff reduced the total volume to 0.6 TB. It took a bit over 11 days until the TM exhausted its 500 GB disk budget for the first time and started to expire data. The NIDS reported 66,000 operator-level notifications according to the configured policy, with 98% of them referring to scanning activity.

5.2 NIDS Controls The TM

The TM provides a network-accessible control interface that the NIDS can use to dynamically change operating parameters based on its analysis results such as cutoffs, buffer budgets, and timeouts. In our installation, we instrument the NIDS so that for every operator notification[5], it instructs the TM to *(i)* disable the cutoff for the affected connection for non-scan notifications, and *(ii)* change the storage class of the IP address the attacker is coming from to a more conservative set of parameters (higher cutoffs, longer timeouts), and also assign it to separate memory and buffer pools. The latter significantly increases the retention time for the host's activity, as it now no longer shares its buffer space with the much more populous benign traffic.

[5]We note that the specifics of what constitutes an operator notification vary from site to site, but because we cannot report details of LBNL's operational policy we will refer only to broad *classes* of notifications such as "scans".

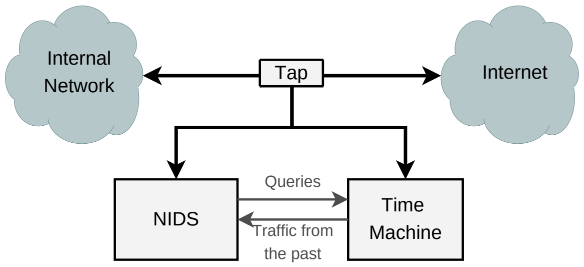


Figure 11: Coupling TM and NIDS at LBNL.

In concrete terms, we introduce two new TM storage classes: *scanners*, for hosts identified as scanners, and *alarms*, for hosts triggering operator notifications other than scan reports. The motivation for this separation is the predominance of Internet-wide scanning: in many environments, scanning alerts heavily dominate the reporting. By creating a separate buffer for scanners, we increase the retention time for notifications not related to such activity, which are likely to be more valuable. The classes *scanners* and *alarms* are provided with a memory budget of 75 MB and a disk budget of 50 GB each. For scanners, we increase the cutoff from 15 KB to 50 KB; for all other offenders we disable the cutoff altogether. Now, whenever the NIDS reports an operator notification, it first sends a `suspend_cutoff` command for the triggering connection to the TM. It then issues a `set_class` command for the offending host, putting the address into either *scanners* or *alarms*.

Examining the commands issued by the NIDS during the two-week period, we find that it sent 427 commands to suspend the cutoff for individual connections. Moreover, it moved 12,532 IP addresses into the *scanners* storage class and 592 into the *alarms* storage class.[6]

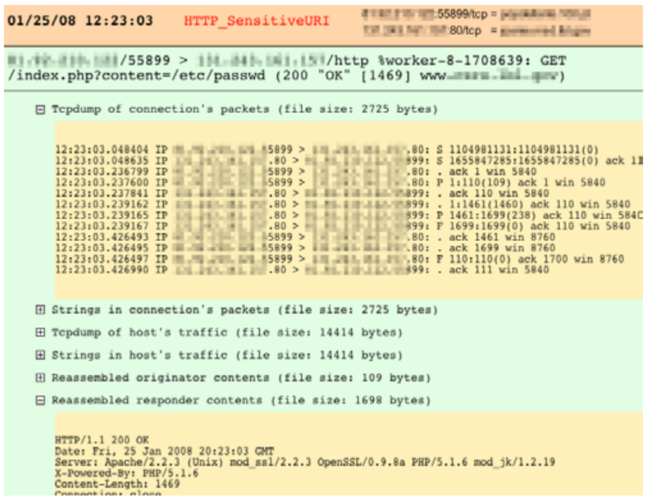
5.3 NIDS Retrieves Data From TM

Another building block for better forensics support is automatic preservation of incident-related traffic. For all operator notifications in our installation, the NIDS queries the TM for the relevant packets, which are then permanently stored for later inspection.

Storage: The NIDS issues up to three queries for each major (non-scan) notification. Two `to_file` queries instruct the TM to store *(i)* all packets of the relevant connection and *(ii)* all packets involving the offender's IP address within the preceding hour. For TCP traffic, the NIDS issues a `feed` query asking it to also *return* the connection's packets to the NIDS. The NIDS then stores the *reassembled* payload stream on disk. For many application protocols, this eases subsequent manual inspection of the activity. We restrict connection queries to in-memory data, while host queries include disk-buffered traffic as well. Our motivation is that connection queries are time-critical while host queries are related to forensics.

During the examined two-week period, the NIDS issued queries for 427 connections (after duplicate elimination) and 376 individual hosts. As queries for connections were limited to in-memory data, their mean processing time was 210 ms (σ= 510 ms). Among the queries, there was one strong outlier that took 10.74 sec to com-

[6]We note that the number of issued commands does not directly correspond to the number of operator notifications generated by the NIDS. The NIDS often reports hosts and connections multiple times, but only sends the corresponding command once. Furthermore, the NIDS sometimes issues commands to change the storage class for activity which does not generate a notification.



```
01/25/08 12:23:03     HTTP_SensitiveURI     [redacted]:55899/tcp = [redacted]
                                            [redacted]:80/tcp  = [redacted]

[redacted]/55899 > [redacted]/http %worker-8-1708639: GET
/index.php?content=/etc/passwd (200 "OK" [1469] www[redacted])

⊟ Tcpdump of connection's packets (file size: 2725 bytes)

    12:23:03.048404 IP [redacted]5899 > [redacted].80: S 1104981131:1104981131(0)
    12:23:03.048635 IP [redacted].80 > [redacted]899: S 1655847285:1655847285(0) ack 11
    12:23:03.236799 IP [redacted]5899 > [redacted].80: . ack 1 win 5840
    12:23:03.237600 IP [redacted]5899 > [redacted].80: P 1:110(109) ack 1 win 5840
    12:23:03.237841 IP [redacted].80 > [redacted]899: . ack 110 win 5840
    12:23:03.239162 IP [redacted].80 > [redacted]899: . 1:1461(1460) ack 110 win 5840
    12:23:03.239165 IP [redacted].80 > [redacted]899: P 1461:1699(238) ack 110 win 584C
    12:23:03.239167 IP [redacted].80 > [redacted]899: F 1699:1699(0) ack 110 win 5840
    12:23:03.426493 IP [redacted]5899 > [redacted].80: . ack 1461 win 8760
    12:23:03.426495 IP [redacted]5899 > [redacted].80: . ack 1699 win 8760
    12:23:03.426497 IP [redacted]5899 > [redacted].80: F 110:110(0) ack 1700 win 8760
    12:23:03.426990 IP [redacted].80 > [redacted]899: . ack 111 win 5840

⊞ Strings in connection's packets (file size: 2725 bytes)
⊞ Tcpdump of host's traffic (file size: 14414 bytes)
⊞ Strings in host's traffic (file size: 14414 bytes)
⊞ Reassembled originator contents (file size: 109 bytes)
⊟ Reassembled responder contents (file size: 1698 bytes)

    HTTP/1.1 200 OK
    Date: Fri, 25 Jan 2008 20:23:03 GMT
    Server: Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.8a PHP/5.1.6 mod_jk/1.2.19
    X-Powered-By: PHP/5.1.6
    Content-Length: 1469
    Connection: close
```

Figure 12: Web-interface to notifications and their corresponding network traffic (packets and payload).

plete: it yielded 299,002 packets in response. Manual inspection of the extracted traffic showed that this was a large DNS session. Excluding this query, the mean time was 190 ms ($\sigma = 100$ ms). Queries for individual hosts included on-disk data as well, and therefore took significantly longer; 25.7 sec on average. Their processing times also varied more (median 10.2 sec, $\sigma = 54.1$ sec).

Interactive Access: To further reduce the turnaround time between receiving a NIDS notification and inspecting the relevant traffic, we developed a Web-based interface that enables browsing of the data associated with each notification; Fig. 12 shows a snapshot. The prototype interface presents the list of notifications and indicates which kind of automatically extracted TM traffic is available. The operator can then inspect relevant packets and payload using a browser, including traffic that occurred prior to the notification.

Experiences: We have been running the joint TM/NIDS setup at LBNL for two months, and have used the system to both analyze packet traces and reassembled payload streams for more detailed analysis. During this time, the TM has proven to be extremely useful. First, one often just cannot reliably tell the impact of a specific notification without having the actual traffic at hand. Second, it turns out to be an enormous timesaver to always have the traffic related to a notification available for *immediate* analysis. This allows the operator to inspect a significantly larger number of cases in depth than would otherwise be possible, even those that appear to be minor on first sight. Since with the TM/NIDS setup double-checking even likely false-positives comes nearly for free, the overall quality of the security monitoring can be significantly improved.

Our experience from the deployment confirms the utility of such a setup in several ways. First, the TM enables us to assess whether an attack succeeded. For example, a still very common attack includes probing web servers for vulnerabilities. Consider Web requests of the form `foo.php?arg=../../../etc/passwd` with which the attacker tries to trick a CGI script into returning a list of passwords. Since many attackers scan the Internet for vulnerable servers, simply flagging such requests generates a large number false positives, since they very rarely succeed. If the NIDS reports the server's response code, the operator can quickly weed out the cases where the server just returned an error message. However, even when the server returns an *200 OK*, this does *not* necessarily indicate a successful attack. Often the response is instead a generic, harmless page (e.g., nicely formatted HTML explaining that the request was invalid). Since the TM provides the served web page in

```
XXX.XXX.XXX.XXX/57340 > XXX.XXX.XXX.XXX/smtp same gap on
     link/time-machine (> 124/6296)
XXX.XXX.XXX.XXX/55529 > XXX.XXX.XXX.XXX/spop same gap on
     link/time-machine (> 275/165)
XXX.XXX.XXX.XXX/2050  > XXX.XXX.XXX.XXX/pop-3 same gap on
     link/time-machine (> 17/14)
```

Figure 13: Example of drops confirmed by the TM.

its raw form, we can now quickly eliminate these as well. To further automate this analysis, we plan to extend the setup so that the NIDS *itself* checks the TM's response for signs of an actual password list, and suppresses the notification unless it sees one. Similar approaches are applicable to a wide range of probing attacks.

For applications running on non-standard ports the TM has the potential to significantly help with weeding out false-positives. Bro, for example, flags outgoing packets with a destination port 69/udp as potential "Outbound TFTP" (it does not currently include a TFTP protocol analyzer). Assessing the significance of this notification requires looking at the payload. With the TM recordings we were able to quickly identify in several instances that the reported connection reflected BitTorrent traffic rather than TFTP. In another case, Bro reported parsing errors for IRC traffic on 6667/tcp; inspection of the payload quickly revealed that a custom protocol was using the port.

The information captured by the TM can also shed light on how attacks work. In one instance, a local client downloaded a trojan via HTTP. The NIDS reported the fact and instructed the TM to return the corresponding traffic. Once the NIDS had reassembled the payload stream, the trojan's binary code was available on disk for further manual inspection (though truncated at the 15 KB cutoff).

Finally, the TM facilitates the extraction of packet traces for various interesting network situations, even those not necessarily reflecting attacks. Among others, we collected traces of TCP connections opened simultaneously by both sides; sudden FIN storms of apparently misconfigured clients; and packets that triggered inaccuracies in Bro's protocol processing.

5.4 Retrospective Analysis

In the following, we demonstrate the potential of a tighter integration of TM and NIDS by examining forms of *retrospective analysis* this enables.

Recovering from Packet Drops: Under heavy load, a NIDS can lack the processing power to capture and analyze the full packet stream, in which case it will incur *measurement drops* [10]. Working in conjunction with the TM, however, a NIDS can query for connections that are missing packets and reprocess them. If the same gap also occurs in the response received from the TM, the NIDS knows that most likely the problem arose external to the NIDS device (e.g., in an optical tap shared by the two systems, or due to asymmetric routing).

We implemented this recovery scheme for the Bro NIDS. With TCP connections, Bro infers a packet missing if it observes a sequence gap purportedly covered by a TCP acknowledgment. In such cases we modified Bro to request the affected connection from the TM. If the TM connection is complete, Bro has recovered from the gap and proceeds with its analysis. If the TM connection is however also missing the packet, Bro generates a notification (see Fig. 13). In addition to allowing Bro to correctly analyze the traffic that it missed, this also enables Bro to differentiate between drops due to overload and packets indeed missing on the link.

Offloading the NIDS: NIDS face fundamental trade-offs between depth of analysis and resource usage [24]. In a high-volume environment, the operator must often choose to forego classes of analysis due to limited processing power. However, by drawing upon

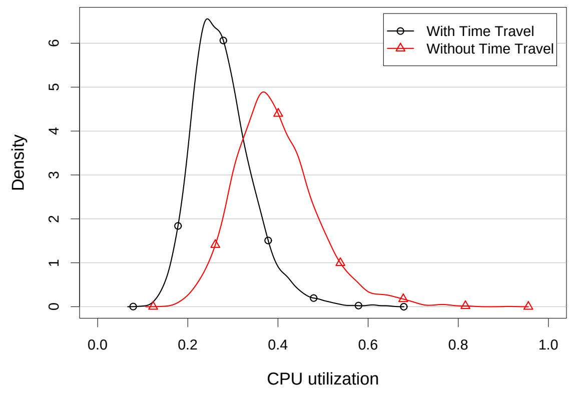


Figure 14: CPU load with and without Time Travel.

the TM, a NIDS can make fine-grained exceptions to what would otherwise be analysis omissions. It does so by requesting initially excluded data once the NIDS recognizes its relevance because of some related analysis that is still enabled.

For example, the bulk of HTTP traffic volume in general originates from HTTP servers, rather than clients. Thus, we can significantly offload a NIDS by restricting its analysis to client-side traffic, i.e., only examine URLs and headers in browser requests, but not the headers and items in server replies. However, once the NIDS observes a suspicious request, it can query the TM for the complete HTTP connection, which it then analyzes with full server-side analysis. The benefit of this setup is that the NIDS can now save significant CPU time as compared to analyzing *all* HTTP connections, yet sacrificing little in the way of detection quality.

FTP data transfers and portmapper activity provide similar examples. Both of these involve dynamically negotiated secondary connections, which the NIDS can discern by analyzing the (lightweight) setup activity. However, because these connections can appear on arbitrary ports, the NIDS can only inspect them directly if it foregoes port-level packet filtering. With the TM, however, the NIDS can request subscriptions (§3.2) to the secondary connections and inspect them in full, optionally also removing the cutoff if it wishes to ensure that it sees the entire contents.

We explore the HTTP scenario in more detail to understand the degree to which a NIDS benefits from offloading some of its processing to the TM. For our assessment, we need to compare two different NIDS configurations (with and without the TM) while processing the same input. Thus, we employ a trace-based evaluation using a 75 min full-HTTP trace captured on LBNL's upstream link (21 GB; 900,000 HTTP sessions), using a two-machine setup similar to that in §4.2. The evaluation requires care since the setup involves communication with the TM: when working offline on a trace, both the NIDS and the TM can process their input more quickly than real-time, i.e., they can consume 1 sec worth of measured traffic in less than 1 sec of execution time. However, the NIDS and the TM differ in the rate at which they outpace network-time, which can lead to a desynchronization between them.

To address these issues, the Bro system provides a *pseudo-realtime* mode [25]: when enabled, it inserts delays into its execution to match the inter-packet gaps observed in a trace. When using this mode, Bro issues queries at the same time intervals as it would during live execution. Our TM implementation does not provide a similar facility. However, for this evaluation we wish to assess the NIDS's operation, rather than the TM's, and it therefore suffices to ensure that the TM correctly replies to all queries. To achieve this, we preload the TM with just the relevant subset of the trace, i.e., the small fraction of the traffic that the Bro NIDS will request from the TM. The key for preloading the TM is predicting which connections the NIDS will request. While in practice the NIDS would trigger HTTP-related queries based on URL patterns, for our evaluation we use an approach independent of a specific detection mechanism: Bro requests each HTTP connection with a small, fixed probability p.

Our first experiment measures the performance of a stand-alone NIDS. We configure Bro to perform full HTTP processing. To achieve a fair comparison, we modify Bro to ignore all server payload after the first 15 KB of each connection, simulating the TM's cutoff. We then run Bro in pseudo-realtime mode on the trace and log the CPU usage for each 1 sec interval. Fig. 14 shows the resulting probability density.

With the baseline established, we then examine the TM/NIDS hybrid. We configure Bro to use the same configuration as in the previous experiment, except with HTTP response processing disabled. Instead, we configure Bro to issue queries to the TM for a pre-computed subset of the HTTP sessions for complete analysis. We choose $p = 0.01$, a value that from our experience requests full analysis for many more connections than a scheme based on patterns of suspicious URLs would. We supply Bro with a prefiltered version of the full HTTP trace with all server-side HTTP payload packets excluded.[7] As described above, we provide the TM with the traffic which Bro will request.

We verify that the TM/NIDS system matches the results of the stand-alone setup. However, Fig. 14 shows a significant reduction in CPU load. In the stand-alone setup, the mean per-second CPU load runs around 40% (σ= 9%). With TM offloading, the mean CPU load decreases to 28%, (σ= 7%). We conclude that offloading indeed achieves a significant reduction in CPU utilization.

Broadening the analysis context: Finally, with a TM a NIDS can request historic network traffic, allowing it to perform analysis on past traffic within a context not available when the traffic originally appeared. For example, once the NIDS identifies a source as a scanner, it is prudent to examine *all* of its traffic in-depth, including its previous activity. The same holds for a local host that shows signs of a possible compromise. Such an in-depth analysis may for example include analyzers that were previously disabled due to their performance overhead. In this way the NIDS can construct for the analyst a detailed application-level record of the offender, or the NIDS might itself assess this broader record against a meta-policy to determine whether the larger view merits an operator notification.

5.5 Implementing Retrospective Analysis

Implementing the TM/NIDS interface for the above experiments requires solving a number of problems. The main challenge lies in that processing traffic from the past, rather than freshly captured, violates a number of assumptions a NIDS typically makes about packets appearing in real-time with a causal order reflecting a monotonic passage of time.

A simple option is to special-case the analysis of resurrected packets by introducing a second data path into the NIDS exclusively dedicated to examining TM responses. However, such an approach severely limits the power of the hybrid system, as we in this case cannot leverage the extensive set of tools the NIDS already provides for live processing. For example, offloading applications, as described in §5.4, would be impossible to realize without duplicating much of the existing code. Therefore, our main design ob-

[7]We prefilter the trace, rather than installing a Bro-level BPF filter, because in a live setting the filtering is done by the kernel, and thus not accounted towards the CPU usage of the Bro process.

jective for our Bro implementation is to process all TM-provided traffic inside the NIDS's *standard* processing path, the same as for any live traffic—and in parallel with live traffic. In the remainder of this section, we discuss the issues that arose when adding such a TM interface to the Bro NIDS.

Bro Implementation: Bro provides an extensive, domain-specific scripting language. We extend the language with a set of predefined functions to control and query the TM, mirroring the functionality accessible via the TM's remote interface (see §3.2), such as changing the TM class associated with a suspect IP address, or querying for packets based on IP addresses or connection 4-tuples. One basic requirement for this is that the interface to the TM operates asynchronously, i.e., Bro must not block waiting for a response.

Sending commands to the TM is straight-forward and thus omitted. Receiving packets from the TM for processing, however, raises subtle implementation issues: the timestamp to associate with received query packets, and how to process them if they are replicates of ones the NIDS has already processed due to direct capture from the network, or because the same packet matches multiple streams returned for several different concurrent queries.

Regarding timestamps, retrieved packets include the time when the TM recorded them. However, this time is in the past and if the NIDS uses it directly, confusion arises due to its assumptions regarding time monotonicity. For example, Bro derives its measure of time from the timestamps of the captured packets. For example it uses these timestamps to compute timer expirations and to manage state. The simple solution of rewriting the timestamps to reflect the current time confounds any analysis that relies on either absolute time or on relative time between multiple connections. Such an approach also has the potential to confuse the analyst that inspects any timestamped or logged information.

The key insight for our solution, which enables us to integrate the TM interface into Bro with minimal surgery, is to restrict Bro to always request *complete connections* from the TM rather than individual packets. Such a constraint is tenable because, like all major NIDS, connections form Bro's main unit of analysis.

We implement this constraint by ensuring that Bro only issues queries in one of two forms: *(i)* for all packets with the same 4-tuple $(\text{address}_1, \text{port}_1, \text{address}_2, \text{port}_2)$, or *(ii)* for all packets involving a particular address. In addition, to ensure that Bro receives all packets for these connections, including future ones, it *subscribes* to the query (see §3.2).

Relying on complete connections simplifies the problem of timestamps by allowing us to introduce the use of *per-query network times*: for each TM query, Bro tracks the most recently received packet in response to the query and then maintains separate per-query *timelines* to drive the management of any timer whose instantiation stems from a retrieved packet. Thus, TM packets do not perturb Bro's global timeline (which it continues to derive from the timestamps of packets in its direct input stream).

We also rely on complete connections to address the issue of replicated input. When retrieved packets for a connection begin to arrive while Bro is processing the same connection via its live feed, it discards the live version and starts afresh with the TM version. (It also discards any future live packets for such connections, since these will arrive via its TM subscription.) Moreover, if Bro is processing packets of a connection via the TM and then receives packets for this same connection via its live feed (unlikely, but not impossible if the system's packet capturing uses large buffers), then Bro again ignores the live version. Finally, if Bro receives a connection multiple times from the TM (e.g., because of multiple matching queries), it only analyzes the first instance.

Our modifications to Bro provide the NIDS with a powerful interface to the TM that supports forensics as well as automatic, retrospective analysis. The additions introduce minimal overhead, and have no impact on Bro's performance when it runs without a TM.

6. DEPLOYMENT TRADE-OFFS

In an actual deployment, the TM operator faces several trade-offs in terms of CPU, memory, and disk requirements. The most obvious trade-off is the design decision of foregoing complete storage of high-volume connections in order to reduce memory/disk consumption. There are others as well, however.

Risk of Evasion: The TM's cutoff mechanism faces an obvious risk for evasion: if an attacker delays his attack to occur after the cutoff, the TM will not record the malicious actions. This is a fundamental limitation of our approach. However, short of comprehensively storing *all* packets, any volume reduction heuristic faces such a blind spot.

The cutoff evasion problem is similar in risks to the problem NIDS face when relying on timeouts for state management. If a multi-step attack is stretched over a long enough time period such that the NIDS is forced to expire its state in the interim the attack can go undetected. Yet, to avoid memory exhaustion state must be expired eventually. Therefore, NIDS rely on the fact that an attacker cannot predict when *exactly* a timeout will take place [10].

Similarly, the TM has several ways for reducing the risk of evasion by making the cutoff mechanism less predictable. One approach is to use different storage classes (see §3.1) with different cutoffs for different types of traffic, e.g., based on applications (for some services, delaying an attack to later stages of a session is harder than for others). As discussed in §5.2, we can also leverage a NIDS's risk assessment to dynamically adjust the cutoff for traffic found more likely to pose a threat. Finally, we plan to examine *randomizing* the cutoff so that *(i)* an attacker cannot predict at which point it will go into effect, and *(ii)* even when the cutoff has been triggered, the TM may continue recording a random subset of subsequent packets.

Network Load: When running in high-volume 10 Gbps environments, the TM can exceed the limits of what commodity hardware can support in terms of packet-capture and disk utilization. We can alleviate this impact with use of more expensive, special-purpose hardware (such as the Endace monitoring card at MWN), but at added cost and for limited benefit. We note, however, that the TM is well-suited for *clustering* in the same way as a NIDS [26]: we can deploy a set of PCs, each running a separate TM on a slice of the total traffic. In such a distributed setting, an additional front-end system can create the impression to the user of interacting with a single TM by relaying to/from all backend TMs.

Floods: Another trade-off concerns packet floods, such as encountered during high-volume DoS attacks. Distributed floods stress the TM's connection-handling, and can thus undermine the capture of useful traffic. For example, during normal operation at MWN an average of 500,000 connections are active and stored in the TM's connection table. However, we have experienced floods during which the number of connections increased to 3–4 million within 30 seconds. Tracking these induced massive packet drops and eventually exhausted the machine's physical memory.

In addition, adversaries could attack the TM directly by exploiting its specific mechanisms. They could for example generate large numbers of small connections in order to significantly reduce retention time. However, such attacks require the attacker to commit significant resources, which, like other floods, will render them vulnerable to detection.

To mitigate the impact of floods on the TM's processing, we plan to augment the TM with a flood detection and mitigation mechanism. For example, the system can probabilistically track per-source thresholds of connection attempts and resort to address-specific packet sampling once a source exceeds these. Alternatively, when operating in conjunction with a NIDS that includes a flood detection mechanism, the TM can rely upon the NIDS to decide when and how the TM should react.

Retrieval Time: When running a joint TM/NIDS setup, we need to consider a trade-off between the response time for answering a query versus the time range that the TM examines to find the relevant packets. As discussed in §4.2, the TM can answer queries quite quickly as long as it restricts its retrieval to in-memory data. However, once the TM needs to search its disk, queries can take seconds to minutes, even if they include a time interval to limit the scope of the search. Thus, the NIDS must issue such queries carefully so as to not exhaust the TM's resources. We do not yet have enough long-term operational experience to have determined good rules for how to manage such queries, but this is part of the near-term focus of our future work.

NIDS and Cutoff: A final issue concerns the impact of the TM's cutoff on the NIDS processing. In our NIDS implementation, we minimize the limitations resulting from the cutoff by combining each NIDS query with a request to remove the cutoff for the associated connections or addresses. This takes care of future activity via the TM. But there is little we can do about flows curtailed in the past—those for which the TM already applied the cutoff. Recall that the general premise of the TM is that we usually can operate the TM with a sufficiently large cutoff that information of interest is captured. To further reduce this problem the TM always stores all TCP control packets (SYN/FIN/RST), thus enabling a NIDS to perform its connection-level analysis.

7. RELATED WORK

In this paper we develop a "Time Machine" for efficient network packet recording and retrieval, and couple the resulting system with a NIDS. The basic approach is to leverage the heavy-tailed nature of Internet traffic [17, 19] to significantly reduce the volume of bulk traffic recording. Our work builds extensively on the proof-of-principle prototype we describe in [15], greatly increasing its performance and coupling it to external systems to support automated querying, live connections to a NIDS, and "subscriptions" to future packets satisfying a given query. These features have enabled us to then use the system in conjunction with the Bro NIDS in an operational high-volume setting.

Different approaches have been suggested in the literature to record high-volume network traffic. First, several systems aim to record full packet traces: Anderson et al. [1] records at kernel-level to provide bulk capture at high rates, and Antonelli et al. [2] focuses on long-term archive and stores traffic on *tapes*. However, these systems do not provide automated and efficient real-time query interfaces. Hyperion [9] employs a dedicated stream file system to store high-volume data streams, indexing stream data using Bloom filters. Hyperion bulk-records entire traffic streams, and does not a provide features for automatic or semi-automatic forensics nor coupling with a NIDS. Gigascope [8], on the other hand, supports SQL-like queries on a packet stream, but no long-term archiving.

Another approach is to store higher-level abstractions of the network traffic to reduce the data volume: Reiss et al. [21] record flows and provide real-time query facilities; Shanmugasundaram et al. [23] record key events of activity such as connections and scans; and [3, 16] both provide frameworks suitable for performing different data reduction techniques. ([16] is based on the CoMo platform [6]). Cooke et al. [7] aggregate data as it ages: first packets are stored; these are than transformed into flows. They focus on storage management algorithms to divide storage between the different aggregation levels. Ponec et al. [20] store traffic digests for payload attribution; the queries do not yield the actual content. Any data reduction decreases the amount of information available. We argue that for security applications, the TM's approach of archiving the head of connections at the packet-level provides an attractive degree of detail compared to such abstractions.

Reducing traffic volume by omitting parts of the traffic is employed by the Shunt [13]. The Shunt is a programmable NIC that an associated NIDS/NIPS instructs to forward, drop, or *divert* packets at a per-connection granularity. The drop functionality supports intrusion prevention; the forward functionality supports offloading the NIDS for streams it has determined it will forego analyzing; and the divert functionality allows the NIDS to inspect and potentially intercede any traffic it wishes. The TM could leverage the Shunt to impose the cutoff directly on the NIC.

While intrusion detection systems such as Snort [22] and Bro [18] can record traffic, they typically keep only a small subset of the network's packets; for Snort, just those that triggered an alert, and for Bro just those that the system selected for analysis. Neither system—nor any other of which we are aware—can incorporate network traffic recorded in the past into their *live* analysis. We added this capability to the Bro system.

Commercial vendors, e.g., [4, 14, 12], offer a number of packet recorders. Due to their closed nature, it is difficult to construct a clear picture of their capabilities and performances. As far as we can tell, none of these has been coupled with a NIDS.

Finally, the notion of "time travel" has been discussed in other contexts of computer forensics. For instance, ReVirt [11] can reconstruct past states of a virtual machine at the instruction-level.

8. CONCLUSION

In this work we explore the significant capabilities attainable for network security analysis via *Time Travel*, i.e., the ability to quickly access past network traffic for network analysis and security forensics. This approach is particular powerful when integrating traffic from the past with a real-time NIDS's analysis. We support Time Travel via the Time Machine (TM) system, which stores network traffic in its most detailed form, i.e., as packets. The TM provides a remote control-and-query interface to automatically request stored packets and to dynamically adapt the TM's operation parameters. To reduce the amount of traffic stored, the TM leverages a simple but effective "cutoff" heuristic: it only stores the first N bytes of each connection (typically, N = 10–20 KB). This approach leverages the heavy-tailed nature of network traffic to capture the great majority of connections in their entirety, while omitting storage of the vast majority of the bytes in a traffic stream.

We show that the TM allows us to buffer most connections completely for minutes *in memory*, and on disk for *days*, even in 10 Gbps network environments, using only commodity hardware. The cutoff heuristic reduces the amount of data to store to less than 10% of the original traffic. We add TM support to the open-source Bro NIDS, and examined a number of applications (controlling the TM, correlating NIDS alarms with associated packet data, and retrospective analysis) that such integration enables. In addition, we explore the technical subtleties that arise when injecting recorded network traffic into a NIDS that is simultaneously analyzing live traffic. Our evaluation using traces as well as live traffic from two large sites finds that the combined system can process up to 120 retrospective queries per second, and can potentially analyze traffic seen 4–15 days in the past, using affordable memory and disk resources.

Our previous proof-of-principle TM implementation has been in operational use for several years at LBNL. The new, joint TM/NIDS installation is now running there continuously in a prototype setup, and the site's operators are planning to integrate it into their operational security monitoring.

To further improve performance in high-volume environments, we plan to develop a version of the system that implements cutoff processing in dedicated hardware (such as the Shunt FPGA [13]) or in the kernel, in order to reduce the traffic volume as early as possible. Using ideas from [7], we also plan to further extend the period we can "travel back in time" by aggregating packet data into higher level representations (e.g., flows) once evicted from the TM's buffers.

Overall, we have found that retrospective analysis requires a great deal of experience with a TM/NIDS setup in operational environments to identify the most useful applications, especially considering the trade-offs discussed in §6. Now that we have the TM/NIDS hybrid in place, the next step is to pursue a study of these possibilities.

9. ACKNOWLEDGMENTS

We would like to thank Stefan Kornexl for implementing the earlier TM prototype, Christian Kreibich for helping with the Broccoli interface, Jim Mellander for sharing his experiences with the TM's deployment, and the anonymous reviewers for their valuable comments. We would also like to thank the Lawrence Berkeley National Laboratory; the Leibniz-Rechenzentrum, München; and the University of California, Berkeley. This work was supported in part by NSF Awards STI-0334088, NSF-0433702, CNS-0627320, CNS-0716640, CNS-0722035, a grant from the Bavaria California Technology Center, and a grant from Deutsche Telekom Laboratories, Berlin. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators and do not necessarily reflect the views of the National Science Foundation.

10. REFERENCES

[1] ANDERSON, E., AND ARLITT, M. Full Packet Capture and Offline Analysis on 1 and 10 Gb/s Networks. Tech. Rep. HPL-2006-156, HP Labs, 2006.

[2] ANTONELLI, C., CO, K., M FIELDS, AND HONEYMAN, P. Cryptographic Wiretapping at 100 Megabits. In *SPIE 16th Int. Symp. on Aerospace Defense Sensing, Simulation, and Controls.* (2002).

[3] CHANDRASEKARAN, S., AND FRANKLIN, M. Remembrance of Streams Past: Overload-sensitive Management of Archived Streams. In *Proc. Very Large Data Bases* (2004).

[4] ClearSight Networks. `http://www.clearsightnet.com`.

[5] CNET NEWS. Another suspected NASA hacker indicted. `http://www.news.com/2102-7350_3-6140001.html`.

[6] CoMo. `http://como.sourceforge.net`.

[7] COOKE, E., MYRICK, A., RUSEK, D., AND JAHANIAN, F. Resource-aware Multi-format Network Security Data Storage. In *Proc. SIGCOMM LSAD workshop* (2006).

[8] CRANOR, C., JOHNSON, T., AND SPATSCHECK, O. Gigascope: A Stream Database for Network Applications. In *Proc. SIGMOD* (2003).

[9] DESNOYERS, P., AND SHENOY, P. J. Hyperion: High Volume Stream Archival for Retrospective Querying. In *Proc. 2007 USENIX Technical Conf* (2007).

[10] DREGER, H., FELDMANN, A., PAXSON, V., AND SOMMER, R. Operational Experiences with High-Volume Network Intrusion Detection. In *Proc. 11th ACM Conf. on Comp. and Comm. Security* (2004).

[11] DUNLAP, G. W., KING, S. T., CINAR, S., BASRAI, M. A., AND CHEN, P. M. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. In *Proc. Symp. on Operating Systems Design and Implementation* (2002).

[12] ENDACE MEASUREMENT SYSTEMS. `http://www.endace.com/`, 2008.

[13] GONZALEZ, J. M., PAXSON, V., AND WEAVER, N. Shunting: A Hardware/Software Architecture for Flexible, High-performance Network Intrusion Prevention. In *Proc. 14th ACM Conf. on Comp. and Comm. Security* (2007).

[14] Intelica Networks. `http://www.intelicanetworks.com`.

[15] KORNEXL, S., PAXSON, V., DREGER, H., FELDMANN, A., AND SOMMER, R. Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic (Short Paper). In *Proc. ACM SIGCOMM IMC* (2005).

[16] MCGRATH, K. P., AND NELSON, J. Monitoring & Forensic Analysis for Wireless Networks. In *Proc. Conf. on Internet Surveillance and Protection* (2006).

[17] PARK, K., KIM, G., AND CROVELLA, M. On the Relationship Between File Sizes, Transport Protocols, and Self-similar Network Traffic. In *Proc. ICNP '96* (1996).

[18] PAXSON, V. Bro: A System for Detecting Network Intruders in Real-Time. *Comp. Networks 31*, 23–24 (1999).

[19] PAXSON, V., AND FLOYD, S. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking 3*, 3 (1995).

[20] PONEC, M., GIURA, P., BRÖNNIMANN, H., AND WEIN, J. Highly Efficient Techniques for Network Forensics. In *Proc. 14th ACM Conf. on Comp. and Comm. Security* (2007).

[21] REISS, F., STOCKINGER, K., WU, K., SHOSHANI, A., AND HELLERSTEIN, J. M. Enabling Real-Time Querying of Live and Historical Stream Data. In *Proc. Statistical & Scientific Database Management* (2007).

[22] ROESCH, M. Snort – Lightweight Intrusion Detection for Networks. In *Proc. 13th Systems Administration Conference - LISA '99* (1999), pp. 229–238.

[23] SHANMUGASUNDARAM, K., MEMON, N., SAVANT, A., AND BRÖNNIMANN, H. ForNet: A Distributed Forensics Network. In *Proc. Workshop on Math. Methods, Models and Architectures for Comp. Networks Security* (2003).

[24] SOMMER, R. *Viable Network Intrusion Detection in High-Performance Environments*. PhD thesis, TU München, 2005.

[25] SOMMER, R., AND PAXSON, V. Exploiting Independent State For Network Intrusion Detection. In *Proc. Computer Security Applications Conf.* (2005).

[26] VALLENTIN, M., SOMMER, R., LEE, J., LERES, C., PAXSON, V., AND TIERNEY, B. The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware. In *Proc. 10th Int. Symp. Recent Advances in Intrusion Detection (RAID)* (2007).

[27] WALLERICH, J., DREGER, H., FELDMANN, A., KRISHNAMURTHY, B., AND WILLINGER, W. A Methodology for Studying Persistency Aspects of Internet Flows. *ACM SIGCOMM CCR 35*, 2 (Apr 2005), 23–36.