Towards Methodical Calibration: A Case Study of Enterprise Switch Measurements

Boris Nechaev[§], Vern Paxson[*ɣ ǂ], Mark Allman[*], Mike Bennett[ǂ], and Andrei Gurtov[§]

TR-13-005

September 2013

Abstract

In this work we discuss the general problem of how to undertake the thorough calibration of empirical data, by which we mean identifying (and ideally remedying) shortcomings and biases present in the data due to the process by which we collected it. We illustrate a methodology for proceeding with such calibration in the context of network trace measurements; in particular, traces captured from switches within an enterprise. We argue that such calibration fundamentally requires proceeding in a progressive fashion, building up an understanding of the data's quality first regarding basic properties and then onward to more complex properties. In addition, the procedure often has an iterative nature, where the investigation of these more complex properties can lead to revisiting earlier calibration steps in order to further refine them. While the methodology often proves labor-intensive, it arguably plays a vital role in establishing the ultimate soundness of any subsequent analysis based on the data.

§ Helsinki Institute for Information Technology HIIT / Aalto University, Teknikvägen 14, 02150 Espoo, Finland
* International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, California, 94704
ɣ University of California, Berkeley, 2200 University Drive, Berkeley, California 94720
ǂ Lawrence Berkeley National Lab, 1 Cyclotron Rd, Berkeley, California, 94720

Many thanks to Nat Stoddard for his extensive efforts capturing the LBL2 dataset. Our thanks too to Scott Campbell for his assistance with VLAN-related trace manipulation. This work was funded in part by NSF grants CNS-0831535 and NeTS-1161799. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

I. Introduction

Any empirical investigation necessarily begins with collecting measurements. The calibration of this data—by which we mean identifying (and ideally remedying) shortcomings and biases present in the measurements due to the process by which we collected them—forms an integral part of data-driven scientific study, since data quality has crucial implications for the soundness of any empirical analysis results. This observation applies equally well to network traffic analysis as to high-energy physics, yet we find little guidance in the network research literature regarding how to actually pursue this undertaking.

In this work we attempt to frame a general methodology for calibrating measurement data. Our approach stems from our extensive experience and long-term interest in conducting network measurement studies [15], and we illustrate the process using a case-study drawn from that domain. However, we believe the overall framework likely has applicability to other data-driven disciplines.

In high-level terms, we advocate for:

1) If possible, build calibration-oriented monitoring into the measurement process itself.
2) Proceed with calibration in a *progressive* fashion, starting with establishing the most basic properties of the data, and then moving on to the next set of properties that we can investigate given our understanding of those basic properties; and so on.
3) At each stage, comprehensively assess properties of the data in terms of our *domain knowledge* of which behaviors we believe should or should not manifest. Inconsistencies in this regard often point up to measurement-related issues, though we must take care to avoid deeming a true phenomenon as a measurement artifact.
4) Be prepared to *iterate*, revisiting the assessment process of previous stages in light of new understandings that emerge from later stages.

In abstract terms, the above procedure aims to enable us to progressively build up confidence regarding our understanding of the data: its peculiarities, skews, omissions, and actual precision/accuracy. When we find problems, in some cases we can perhaps *remedy* them (or at least partially) by applying post-facto corrections. In other cases, we can only note the presence of an issue so that we can take its influence into account when later developing analyses that in part depend on it.

There is of course a world of difference between simply outlining such an approach versus the considerations that actually arise when applying it concretely. To address that concern, we illustrate an application of the above methodology to a specific problem: calibrating network packet trace measurements captured from switches internal to an enterprise.

Our efforts regarding such measurements actually began a number of years ago, with capturing and calibrating enterprise network traces as described in [11]. In that work we discussed a number of calibration steps, but did not illuminate the methodology underlying the process. Subsequent to that effort, we gathered a second extensive set of measurements—monitoring nearly 1,000 hosts—at the same enterprise. For this effort, we drew upon our experiences from calibrating the first set of measurements to modify the capturing procedure. Doing so enabled us to successfully avoid some of the measurement pitfalls we previously discovered—though our revised measuring process also introduced some new issues. In particular, the calibration procedures and results we present here differ significantly from those used in the earlier work (necessarily so, since we changed the measurement procedure precisely to avoid some of the earlier issues). Our new measurements also give us an opportunity to assess the efficacy of some of the previous calibration efforts reported in [11] using new "ground truth" not originally available.

In specific terms, for the traces we address the following calibration issues: measurement-induced loss and reordering, timing fidelity, packet duplication, and inference of network topology. In examining these issues we strive both to demonstrate the nature of our progressive calibration methodology as applied to real data, and to formulate recommendations for sound switch-based trace collection in enterprise environments.

We structure the paper as follows. In § II we describe our capturing apparatus and give an overview of the resulting data. In § III we discuss the first step in the process, building calibration-oriented monitoring into the measurement itself. We then proceed to the post-capture calibration procedure, beginning in § IV with measurement-induced packet reordering and proceeding in § V to measurement-induced loss. § VI frames our efforts to understand network topology from the packet traces, and § VII explores timing fidelity issues. § VIII discusses calibrating packets for which we observe duplicate copies. Finally, we conclude in § IX.

II. Capturing Process and Resulting Data

Our first efforts at calibrating switch-level packet traces, described in detail in [11], focused on a set of measurements taken at the Lawrence Berkeley National Laboratory (LBL) between October 2005 and March 2006. Using the lessons from that effort, we collected

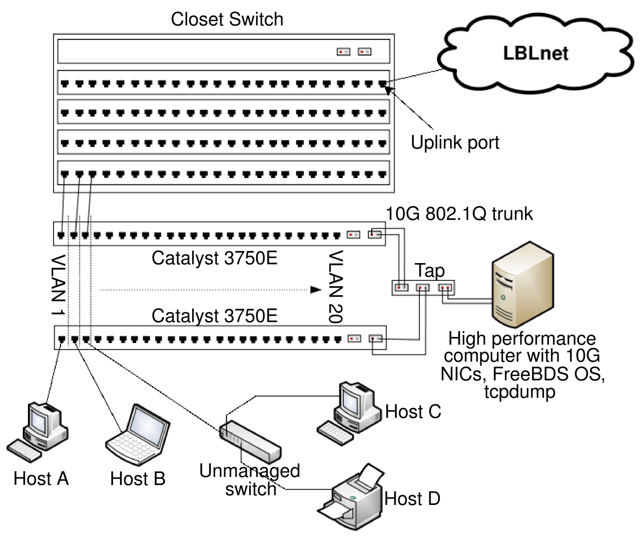

Fig. 1. Measurement apparatus of `LBL2`.

another dataset at LBL from November 2009 through February 2010. The two sets of traces are similar in spirit, but differ in the specifics of the capturing apparatus. In the rest of the paper we refer to the older dataset as `LBL1` and the newer as `LBL2`.

We captured the `LBL1` measurements using passive taps mirroring up to ten 10/100 Mbps Ethernet links (all connected to a single switch). The taps fed into a switch that then aggregated the mirrored traffic onto two 1 Gbps Ethernet links, each carrying the (bidirectional) activity of up to five of the links. We then recorded the full contents of the Gbps links using a `tcpdump` process for each.

As we will describe shortly, this approach introduced some difficulties in interpreting the resulting data, leading us to revise the setup when an opportunity arose to capture new measurements. The capturing apparatus for `LBL2` consisted of two monitoring switches (Catalyst 3750E) with 10 Gbps mirroring ports placed between an LBL closet switch and the monitored end hosts, as depicted in Figure 1. A high performance FreeBSD host running `tcpdump` processes recorded the packets mirrored by the monitoring switch. As in `LBL1`, the production Ethernet cable going from the closet switch could connect myriad devices to the LBL network, e.g., desktop hosts, laptops, printers, or unmanaged switches that in turn connected multiple devices to the network. Each closet switch had an uplink port leading to a router, or to another switch in a chain of switches eventually connecting to a router.

One of the issues we faced when analyzing `LBL1` arose out of the fact that the measurement process grouped together all packets from up to 5 ports into a single trace, for which we had no additional information regarding which port corresponded to which traffic. Such ambiguity makes certain analyses difficult. For instance, consider packet duplicates. In the absence of information associating a given packet with a given port, we lack any direct knowledge of whether a duplicate represents a packet generated by an end host, or whether the switch itself created the copy as part of switched Ethernet's use of flooding. To address this issue in `LBL2`, our capturing apparatus used VLAN tagging [5] of individual ports, with the switch adding a VLAN header with a distinct tag unique to each monitored port, enabling *post facto* association of particular traffic with particular ports.

Another issue complicating the `LBL1` analysis concerned the directionality of a given packet, for which again we lacked any information in the final aggregated trace. For `LBL2`, we remedied this problem by aggregating the two directions of the monitored (and now VLAN-tagged) links onto separate 10 Gbps Ethernet links, recording them via `tcpdump` into two separate trace files. Thus, the particular trace file in which a packet appeared implicitly coded its directionality.

The above changes also provide us with an opportunity to assess the accuracy of the calibration techniques we used for `LBL1` (per § VI and § VIII). Unfortunately, this new strategy also introduced a new measurement-based artifact: packet reordering. Since the different directions of a given traffic flow appeared on different interfaces of the monitoring system, in some instances a later packet arriving on one interface received a timestamp from the kernel before that of an earlier packet arriving on the other interface. Since the timestamps provided the only means for associating sequencing with the packets captured in different traces, such timestamping behavior led to the later packet appearing to have arrived prior to the earlier one. § IV discusses the calibration process that led to discovering and partially remedying this measurement-based artifact.

The LBL network operators periodically moved our monitoring apparatus to a new set of ports on a switch (or a new switch upon exhausting those on the current switch). To maximize representativeness and eliminate possible bias, we selected the monitored switches arbitrarily, with one exception: the operators checked the switch statistics for each chosen port to ensure the port was actually in use.

We had found during our prior `LBL1` analysis that some ports connected to not to a single host but multiple hosts, which required extensive sleuthing and analysis effort to definitively establish. To more clearly understand this situation for the `LBL2` dataset—and to validate the techniques we had previously used to detect such

Stat	LBL1	LBL2
Total time span	1,114 hrs	1,406 hrs
Total volume	400 GB	957 GB
Total number of packets	869M	1625M
Number of days	51	61
Number of traces	100	102
Typical trace duration	23h	23h
Switch ports	∼500	∼1000
Subnets	4	10
IP addresses	332	1471
"Good" internal TCP conns	363K	942K

TABLE I
DATASET CHARACTERISTICS.

unmanaged switches and bridges in LBL1—we also recorded the full switch settings from 175 LBL switches across a 12-day period. Studying the switch logs helped solidify our understanding of the LBL network structure and dynamics. We found particularly useful in this regard the switch CAM tables that store the mappings of MAC addresses connected to particular ports. These tables revealed that 70% of the ports had at most one connected host at a time throughout all 12 days, and for over 94% of these ports, the same host persisted across all 12 days. This finding meant that we can treat ports as *usually* connecting only one host to the switch; but also that the case of connecting multiple hosts occurs frequently enough that we must always bear it in mind.

During the measurement period, LBL used 10, 100 and 1000 Mbps switch ports. Since we selected ports arbitrarily, all three appear in our traces. We used a general strategy of attaching the monitoring apparatus to 10 switch ports and capturing traffic across the set for 24 hours before choosing another set of switch ports and repeating the process. Part way through the data collection, the operators augmented the monitoring apparatus such that we could capture two sets of 10 ports simultaneously, thus doubling the number of ports monitored per day.

In total we captured full payload packet traces encompassing 1,406 hours of network activity across 61 days and spanning more than 1,000 switch ports. The traces contain 1.6 billion packets and 957 GB of payload. Table I summarizes the main characteristics of both the LBL1 and LBL2 trace collections. We use the same definition of "good" TCP connections staying within LBL as in [10]: all properly established connections terminated either by FIN or RST.

III. INCORPORATING CALIBRATION INTO THE MEASURING PROCESS

Ideally we begin planning for calibration even before commencing to gather data. Naturally, we can only undertake such planning if we are already aware of potential pitfalls of the capturing apparatus, and thus it can prove highly valuable to undertake a pilot measurement study, including pursuing subsequent calibration of the recorded data, in order to learn in detail about the nature of the process. However, we know from experience that one can find it difficult to summon the time, energy and resources to thoroughly calibrate preliminary data—particularly if the window-of-opportunity for capturing the main data appears narrow. That said, even a modest degree of calibration effort in this regard can still prove quite illuminating.

A related notion is to perform the initial stages of calibration *during* the capturing process. Here again, even a modest degree of analysis can pay dividends in identifying problems early enough to enable correcting them. We can think of this stage as focused on a basic question: what actually got captured and what did not (even though it should have been). For our case study, in this regard we developed a validation procedure to check each trace file as soon as capturing for it completed. Without such a check, we risked finding ourselves in the unfortunate position of having spent significant operator effort (a resource at a premium) capturing data that itself suffered from significant flaws.

Our validation script reported: (i) the timestamp of the first and last packets in the trace (and hence the duration of the trace), (ii) the maximum packet size (to indirectly ensure the presence of VLAN headers), (iii) the monotonicity of timestamp increases across the trace (to identify coarse clock adjustments), (iv) the presence of duplicate packets (expected to appear due to monitoring multiple switch ports, coupled with the switch's flooding mechanism), (v) the total number of packets in the trace and number of packets per VLAN (checking that the latter summed to the former), (vi) the number of invalid IP, TCP and UDP checksums (to identify layer-1 problems in the capturing setup), (vii) measurement loss, i.e., failures to record some packets (more about this shortly), and ($viii$) the IP and MAC addresses observed in the trace.

Note that these features (other than measurement loss) all had the property of being relatively *cheap* to code up (and compute). An ongoing challenge for calibration efforts concerns just how deeply to go. We advocate a general rule of thumb of: (i) any inexpensive test, even for a very unlikely possibility, is worth developing, and (ii) for more expensive tests, one has to reflect on the cost/benefit tradeoffs to decide whether to implement the test.

The validation script report allowed us to quickly detect problematic trace files, and hence we could either re-take the measurement, or at least discard the trace as suffering from some large measurement-based anomaly. In particular, the report enabled us to identify several instances where the operators failed to switch to a new set of ports, and also to the discovery of a mechanism that could add VLAN headers to the packets twice instead of once (since some maximum packet sizes ranged abnormally high).

Regarding the other report elements, we found that timestamps always increased monotonically, and that duplicate packets appeared only in the downstream direction (i.e., from the switch to the end host), as expected.

We did however find 26 (IP), 9,170 (TCP) and 120 (UDP) instances of incorrect checksums. We note that these values fit reasonably well with the expectations to which domain knowledge led us: IP checksum errors should be rare, since such packets should not travel more than one IP hop (the receiving node discarding the packet due to its invalid checksum); TCP traffic will usually have a high volume that includes disparate wide-area paths, so occasional errors out of millions of packets should not particularly surprise us; UDP traffic will tend to have lower volume and mainly remain in the enterprise, and thus ought to only rarely manifest errors. Such cross-checks with our expectations play a crucial role in the the calibration process: they serve to flag which apparent behaviors merit further investigation. Given the cost (in terms of analyst time) to look into any given behavior, we need to leverage our expectations to perform triage.

For measurement loss, we used a network trace analysis script written in the Bro networking monitoring language [14]. This script works in an end-to-end fashion by assessing how often TCP streams include acknowledgments for data not present in the trace. In the absence of measurement loss, such situations should not arise (assuming the trace includes the start of the corresponding connection), since no well-functioning TCP should ever acknowledge data it has not received.

The script did sometimes report significant measurement loss, up to 5% of TCP packets, which caused us some concern. However, for a number of traces it did not report any loss, and given our experiences with `LBL1` of somewhat comparable measurement loss, we attributed those traces with significant loss rates as likely reflecting high-speed bursts of traffic for which the monitoring apparatus failed to keep up. Interestingly, this was only half of the story, and in fact the script itself was being mislead by the mis-sequencing of packets in the traces. As discussed in the next section, we only identified this phenomenon during our subsequent calibration process, a good illustration of how later stages of calibration can cast earlier stages in a new light.

IV. Measurement Reordering

During the first stage of calibration we have already made a preliminary assessment of *measurement loss*. We looked for cases where our measurement apparatus failed to faithfully capture what actually occurred on the network. Our general technique for finding such loss involves looking for “gaps” in the TCP sequence space where we do not observe the actual data, but do in fact observe that data being acknowledged as having arrived at the destination. We used this approach in [11] and calculated an upper bound of measurement loss to be 2% in the `LBL1` dataset.

We planned to make the assessment of measurement loss for `LBL2` our second calibration stage. In the process of performing the above measurement gap analysis we stumbled upon a dramatic increase in the number of detected gaps. Upon deeper analysis we found that the cause of the excessive number of gaps in the `LBL2` traces is not measurement loss, but rather measurement reordering.

Measurement induced reordering manifests as follows. While we may observe an ACK for some data not yet observed, we find that often the seemingly missing data is in fact present in the trace shortly *after* the ACK. Our collection methodology collects uni-directional traces and then post-facto merges the two directions together based on the recorded timestamps in the traces. Therefore, if one capturing process on the tracing machine gets hung up for a short amount of time an ACK can get timestamped before the corresponding data. This is clearly a measurement problem that can cause havoc when trying to analyze traces to understand TCP dynamics. We therefore set out to fix the traces to the extent possible such that we can undertake TCP and application-layer analysis without further consideration of this measurement-induced artifact.

The discovery of measurement reordering while assessing measurement loss is a good example of the iterative nature of calibration. We haven’t caught a glimpse of measurement reordering during the first calibration stage. Thus, now we were forced to take a step back and instead of calculating measurement loss rate, calibrate a more basic flaw. Coping with measurement reordering substituted measurement loss as the second stage of our calibration effort.

But we note that it is impossible to fully “fix” the traces since we can never recover the true timestamp that should have been associated with a reordered packet. Therefore, while in the end we largely produce a *correct* *ordering* of the packets, we arrive at only an approximation of the timestamps of the reordered packets. Therefore, we must be cautious of the timestamps of these packets in subsequent analysis.

We use the following process to find and fix measurement induced reordering. While we find no duplicate timestamps within a single trace file, when merging bidirectional trace pairs we find that a single timestamp occur for one packet in each trace. This leads to issues later in our process of imposing a correct ordering and we therefore adjusted the timestamps such that each packet in the merged trace has a unique timestamp. Specifically, when we observe two packets with a timestamp of X, we adjust the second packet's timestamp by repeatedly subtracting 1 microsecond—the timestamp granularity in our traces—until the timestamp is unique. At this stage we only care about arriving at a unique set of timestamps and do not worry about the relative order of the segments as we will deal with that with our subsequent processing.

We next divide the packet traces into TCP connections using the IP addresses, port numbers and tracking of the recorded SYNs, FINs and RSTs. We process each connection as the timestamps in the trace suggest the connection transpired, identifying data-carrying packets that their receiver seemingly already acknowledged in the past. Such occurrences reflect measurement artifacts with high likelihood. Simple packet reordering—as observed in previous studies [13], [2]—would not explain this phenomenon, as packet reordering by the network cannot coax an end system to acknowledge a packet it has not received.

Once we identify a measurement-based reordering we swap the positions and the timestamps of the data and acknowledgment segments in the trace. Our experience shows that while the new ordering is better, it is not always correct. We therefore apply the previous two steps iteratively, up to four times. Even after four passes our ordering analysis did not report the traces as fully clean. We find several instances where the end system's TCP behavior is simply strange. This happens in few enough instances that we decided it was better to live with a little imperfection in the resulting traces than to add complexity to the ordering process and possibly introduce subtle issues into the traces.

In `LBL2` we find that our measurement apparatus misordered 6.9 million packets (roughly 0.5%). Haven't we undertaken the mitigating efforts described in this section, all these packets would have been reported as measurement loss gaps. After identifying and fixing these ordering issues as described above, we still found 44 packets that our analysis cannot fix without large additional complexity. In § VII we explore the question of timing fidelity and, among other things, show the distribution of timestamp differences introduced when flipping packets (see Figure 13).

Note: Our heuristic for identifying measurement-based reordering is not perfect. For instance, a measurement loss of a packet X coupled with a spurious retransmission of packet X could also cause our monitor to observe an ACK for (the first) packet X before we observe (the second) packet X. However, given the probability of measurement loss (see § V) and also low instances of spurious retransmissions (e.g., [13], [2] shows TCP's standard RTO-based retransmissions are spurious much less than 1% of the time) we believe the chances of these two events aligning to produce the reordering we observe is quite low. While we cannot fully discount this case, we conclude that the most prevalent form of reordering we find is an artifact of our measurement methodology.

As a final note, TCP's sequence numbers give us a natural way to mitigate the impact of measurement-based reordering in our packet traces. Non-TCP traffic is no doubt also mis-ordered in `LBL2`, but the lack of a general sequencing mechanism means that we cannot fix these issues in a generic fashion. Therefore, when using the data for non-TCP analysis, we need to take into account the possible mis-ordering of packets.

And a general observation here is that sequencing issues may easily appear if two directions of the traffic are recorded by separate processes. If a researcher designs the measurement apparatus this way, she may want to include detection of measurement reordering already in the first stage of calibration.

V. Measurement Loss

With measurement reordering resolved, we can then return to the question of estimating measurement loss based on observations of TCP sequence gaps. For this purpose we use `Bro` [14], which has the capability to reassemble TCP flows. This enables us to detect cases when flows contain ACK packets acknowledging unseen data sequences. Such events unambiguously indicate measurement loss, since only a grievously erroneous TCP would ever acknowledge data that has not arrived. We call such events *sequence gaps*. As we already noted, measurement loss calibration stage rests upon the success of the measurement reordering stage.

We found that the flipping reduced the number of sequence gaps in 72 out of 102 traces, removing 184K gaps amounting to 417MB of payload. Unsurprisingly, we find the difference between the number of sequence gaps before and after the flipping most evident for traces that contain many out-of-order packets. The number of gaps that remain after the flipping, and therefore

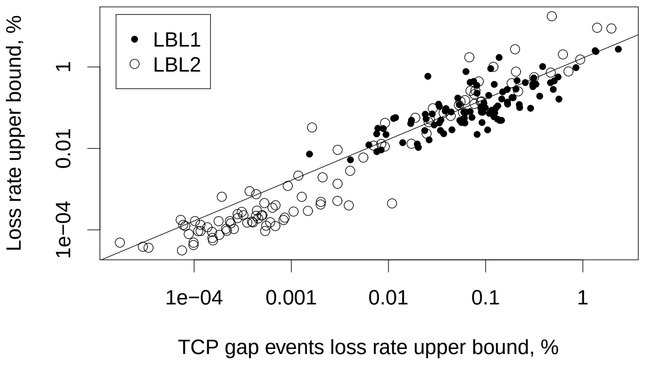

Fig. 2. Estimated measurement loss rates in the `LBL1` and `LBL2` traces. The X-axis gives the rates based on the number of observed gaps in the data acknowledged by TCP receivers. The Y-axis shows the rates estimated from the number of missing bytes reflected by such acknowledgments. The line has the slope 1.

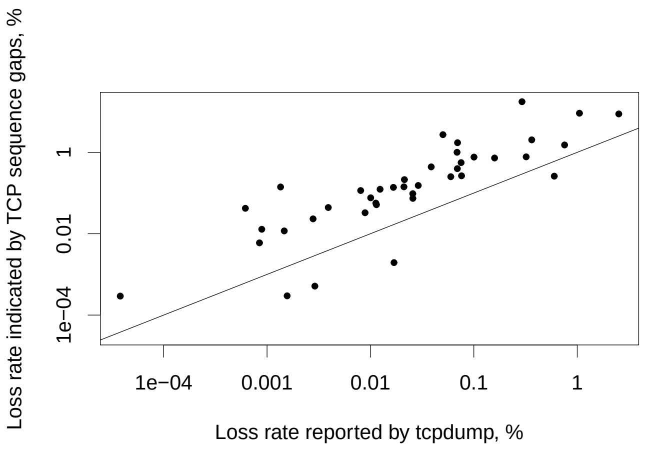

Fig. 3. The `LBL2` measurement loss rates. Y-axis shows the loss rates calculated by the amount of missing bytes. The plot only contains cases when the `tcpdump` loss was non-zero. In the traces where `tcpdump` reported zero loss, the loss rate indicated by TCP sequence gaps remained very small—in all but a pair of cases smaller than 0.001%. The line has slope 1.

represent the true TCP measurement loss, total 272K instances with 1,134MB of payload. Thus, the reordering calibration step removed about 40% of the apparent measurement loss instances, and 27% of the apparent byte-volume of measurement loss.

After calibration, the share of missing bytes reflects 0.12% of the total payload in our traces. Figure 2 compares measurement loss rates observed in `LBL1` and `LBL2` traces. For the majority of traces, we find smaller measurement loss rates for `LBL2` than for `LBL1`, and for both datasets the loss rarely exceeds 1%.

Multiple components in the capturing process can produce measurement loss: capturing switches, taps, the NIC on the apparatus, its kernel, and the `tcpdump` process itself. Without extensive instrumentation, it is hard to estimate the extent to which each of these alternatives contribute to failures to capture the full set of passing network traffic. The only such instrumentation we had available was reports from `tcpdump` indicating the number of packets dropped as reported to it by the kernel. Figure 3 compares the loss rates from these recordings to the ones we deduced by looking at TCP sequence gaps. While our traces hold protocols other than just TCP, and `tcpdump` loss can not account for all the measurement loss we observe, the rough correlation between the two increases our confidence in the validity and representativeness of the results.

VI. Topology

The next aspect of calibration we consider in this paper regards topology: determining the network layout of the monitored site. The elements of topology relevant in switch-based monitoring concern classifying hosts as one of (1) monitored, (2) intra-subnet (same subnet/broadcast domain as the monitor, but not monitored), (3) inter-subnet (different subnet within the enterprise), or (4) external. The internal router defines subnet boundaries by assigning all hosts connected through switches to one port on the router to the same subnet range. External hosts lie outside the enterprise, and all communication between internal and external hosts goes through edge routers.

There is one additional topological entity to consider: “hidden” switches. We might commonly expect that a port of the monitored switch connects to a single end-host. However (as we discussed in [11]) in some cases the port leads to another network switch or hub, perhaps even unbeknownst to the network operators.

Many calibration and analysis efforts rely heavily on meta-information about network topology. In Section VII we used the knowledge of subnet boundaries, which allowed us to distinguish the intra- vs. inter-subnet traffic. This distinction is of importance in enterprise and data center networks, since reaching another subnet requires the packet to traverse a router, which inevitably introduces router queueing and forwarding delays. In data centers, knowledge of topology allows for efficient task allocation by confining the majority of communication within the same rack [3], optimization of bisection bandwidth [1] and design of new routing protocols [16], [17].

We turned our attention to topology calibration when several of our analysis tasks required the knowledge of locality, i.e. whether two hosts belong to the same or different subnets within LBL. For instance, locality is important in timing fidelity calibration that we perform in the next section. As it often happens, our at first mundane topology calibration actions led to a surprising discovery of hidden switches, which are present in both `LBL1` and `LBL2`.

As a first step in calibrating information related to topology, we identify subnet boundaries for all the 102 traces in `LBL2`. Using the IP addresses of the monitored hosts for each trace, we computed the smallest accommodating subnet range. As in [11], the network operators confirmed the accuracy of our results, except for 11 out of 102 traces where the subnet ranges we generated turned out too narrow, primarily because we did not happen to monitor a sufficient number of hosts to cover the whole range.

As a next step in [11] we determined the monitored hosts. This became simple for `LBL2`, since we monitored each traffic direction separately, and we now needed only to extract MAC addresses of sender hosts for the upstream—from the end host to the switch—direction. We couldn't do this for `LBL1`, since we couldn't readily separate upstream and downstream traffic there. Because we can confidently determine monitored hosts in `LBL2`, that ground truth then allows us to assess the accuracy of the graph-coloring algorithm we developed in [11] for the `LBL1` traces. That algorithm relies on building a communication graph between all the observed hosts, and, starting from the router MAC addresses, recursively coloring hosts as `red` or `green`. The `red` class includes routers and represents the non-monitored hosts, while the `green` class yields the monitored hosts. This approach appeared to be accurate when applied to the `LBL1` traces, but we can now assess the accuracy by contrasting its results with the ground truth.

Out of the 1,516 truly monitored MAC addresses in `LBL2`, our approach accurately identifies 1,470 MACs (97%). The algorithm failed to identify 46 MACs as monitored, putting the number of false negatives to 3%. We did not find any false positives, i.e., cases where the script flagged a non-monitored MAC as monitored.

The observed false negatives arise for two reasons. First, our algorithm relies on several thresholds to determine bidirectional communication between two hosts. Those flows that fail to satisfy the thresholds stay uncolored. This can happen if a host has very little or no bidirectional communication with other hosts. Even though we could tweak the thresholds to allow coloring of meager flows, unfortunately in some cases we can't do detection at all. For instance, we observed several truly monitored hosts sending only broadcast packets without ever receiving any packets. In this case no heuristic can determine whether we in face monitor the host. Lowering the thresholds yielded fewer false negatives, but at the price of producing false positives and more coloring inconsistencies.

Further, we found a second peculiar reason for the false negatives. While the algorithm did not produce any coloring inconsistencies for the `LBL1` traces, in `LBL2` we have observed several of them. A coloring inconsistency arises when the algorithm determines to assign a color to a node to which we have already previously assigned the opposite color. We investigated all such cases and found that they all occur due to the host *moving* from the monitored VLAN to a non-monitored VLAN during the time we captured the given trace. This movement means that the host should indeed be eligible for being colored both `red` and `green`. We confirmed that this happens very rarely in our traces—only 4 out of 1,516 monitored hosts exhibited such behavior.

In [11] we faced a major conundrum—we deduced more monitored hosts per trace than the maximum number of capturing taps installed by the network operators. This puzzling observation could have several mundane explanations, but by performing analysis to exclude other potential alternatives, we finally came to the conclusion that in fact some of the monitored switch ports connected to additional switches that had multiple hosts plugged into them. We considered another plausible explanation: that at different times, different end hosts used the same switch port. We refuted it by splitting each trace into 15-minute epochs and finding the maximum number of concurrently active hosts observed within the epoch. We observed that 89 out of 100 `LBL1` traces contained at least one 15-minute period when ***all*** of the deduced monitored hosts appeared together. In the remaining 11 traces only 1 or 2 hosts did not show up together with all the other hosts in the maximum activity epoch. This finding conclusively excluded the above hypothesis.

In general we found applying the scripts developed for the analysis of `LBL1` traces to the `LBL2` traces straightforward—we needed only increase the maximum duplication level from 5 to 10. However, while trying to reproduce the results of the 15-minute epoch analysis described in the previous paragraph, we discovered that the scripts had an implementation bug, which directly affected calculation of the maximum number of concurrently active hosts observed within epoch. Fortunately, fixing the bug changed the results only marginally: the number of concurrently monitored hosts in LBL1 traces became lower by 1 in 11 out of 100 traces. Thus, the conclusions presented in the previous paragraph remain

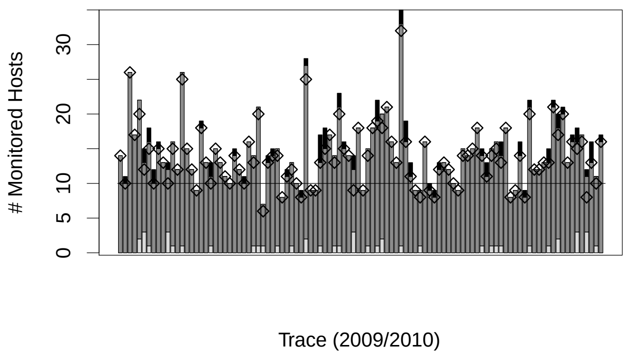

Fig. 4. LBL2 number of monitored hosts.

valid.

We repeated the analysis of the number of monitored hosts per trace for the LBL2 traces. Figure 4 shows the results. In the plot, black represents the true number of monitored hosts as we know from the ground truth. Grey shows the true maximum number of concurrent hosts among all 15-minute epochs. White shows the number of false negatives, i.e., hosts for which our algorithm failed to identify them as monitored. Diamonds represent the number of the maximum concurrent hosts deduced by our script. The deduced number of the maximum concurrent hosts differs from the true number because of false negatives, which arise due to monitored hosts not exhibiting sufficient communication activity for us to discern it reliably or confusing our detection algorithm by moving from the monitored to a non-monitored VLAN. Finally, the line depicts the replication level. In 82% (67% for LBL1) of traces the number of monitored hosts turned out to exceed the number of taps, very strongly suggesting that LBL2 traces also contain hidden switches.

In [11], we confirmed the presence of hidden switches by observing ARP requests between pairs of green hosts not followed by ARP replies, indicating that the hosts could communicate point-to-point unseen, and therefore, while monitored, had a path between them that did not transit our monitoring point. Along these lines, we find 34 out of the 102 LBL2 traces exhibit at least one such pair. At the same time, the coloring analysis for LBL2 identifies that 66 out of the 102 traces (about two thirds) included double-monitored hosts (these did not reside behind a hidden switch). If we assume that in a day long period two hosts connected to the same switch to talk to each other with the probability of two thirds, and the same ratio hold for both monitored and hidden switches, then we can estimate the rough number of traces containing hidden switches. By extrapolating the 34 observed traces, we conclude that in total hidden switches must be present in approximately 51 traces. We find this result to be well in line with the results obtained for LBL1, where a similar extrapolation yielded 42 out 100 traces with hidden switches.

Overall, we conclude that the algorithms developed for topology calibration of the LBL1 traces appears readily applicable to the LBL2 traces. Even without altering their thresholds, they yielded satisfactory results, as supported by comparison with the ground truth.

VII. Timing fidelity

Having completed the topology calibration stage, we can use its results to assess timing fidelity in our dataset. In general, timing fidelity is one of the most commonly discussed calibration aspects due to its importance and prevalence of imperfections associated with it. Timing measurements are strongly tied to the physical constraints of the measurement apparatus, such as clock precision, drift, skew, etc. We have already performed a few basic timing fidelity checks during the first calibration stage. Then we recognized the need for a deeper timing fidelity calibration when in [10] we tried to calculate the bandwidth-delay product of TCP connections.

Packet round-trip time is one of the most important performance metrics. For simple transport protocols such as UDP, it directly affects the user experience. Protocols with sophisticated dynamics such as TCP may exhibit complex behavior depending on the RTT, its variability and stability, and TCP throughput varies in inverse proportion to RTT [12]. More generally, as a basic component of the bandwidth-delay product, RTT plays a crucial role in optimality for many transport protocols. In this section we set out to assess the fidelity and analyze the round-trip times and the one-way delays seen in the LBL1 and LBL2 traces. In [11] we already briefly touched on the matter of timing fidelity by assessing the differences between timestamps of a packet recorded by two network interfaces, though we did not explore the issue in due detail.

In what follows we focus only on TCP packets, since the protocol's specifics provide us with a robust way to discern round-trips based on data sequence and acknowledgment numbers. We performed the TCP flow reassembly using Bro. An important aspect of RTT measurements that directly affects quality and fidelity of results concerns the vantage point position. Ideally, to assess RTT we should measure directly at the host that sends data packets and receives acknowledgment for them. Any other location along the path between a

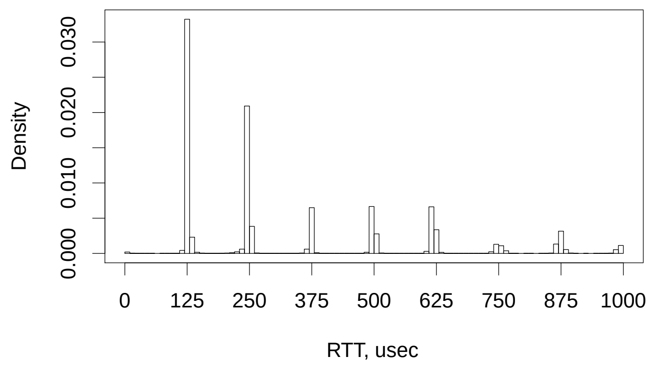

Fig. 5. `LBL1` sub-millisecond intra-subnet handshake RTTs.

sender and a receiver will underestimate RTT values. We can, however, approximate true RTT values by choosing a vantage point sufficiently close to the sender, which for our measurements arises for TCP data packet that originated from a monitored host. Especially for `LBL2`, we know exactly which hosts we have monitored; since we deployed the measurement apparatus between hosts and their attached switches, this gives us the ability to estimate RTTs with a vantage point close to the sender.

When examining TCP flows, we can take advantage of a particular possibility for accurately measuring RTT values. In [6] the authors propose to use TCP's three-way handshake for RTT estimation. Since TCP connection establishment involves three packets—SYN, SYN+ACK, ACK—the time difference between the SYN and its ACK gives the full RTT for the sender/receiver pair. This approach has the very appealing property of working regardless of our vantage point; by leveraging *causality* (that seeing the third packet's arrival will mirror exactly a round-trip having elapsed since seeing the original SYN's arrival), this method yields true RTT values. That said, we need to keep in mind possible packet loss; if, for example, the SYN+ACK packet gets lost, the server will later retransmit it, and the time difference between the SYN and the later ACK will include this (large, and therefore likely noticeable) timeout.

In investigating and characterizing the timing aspects of enterprise network activity, we find it useful to categorize the traffic by locality. Coarsely, we split flows into internal and external, with the former staying within the enterprise boundaries and the latter involving communication outside of the LBL network. Further, to obtain a finer classification, we split the internal traffic into intra- and inter-subnet. Intra-subnet traffic comprises flows confined inside a subnetwork, and inter-subnet communications involve hosts from two subnetworks. Our choice of switches as a vantage point gives us a comprehensive view into intra-subnet behavior. For our purposes the main difference between intra- and inter-subnet traffic is in the latter traversing a router. In untangling various observed timing phenomena we found it handy to be able to exclude the major flow latency component introduced by such complex devices as routers.

Upon beginning our analysis by inspecting the `LBL1` handshake RTTs, we immediately encountered a major puzzle: low RTT values clustered densely around several specific values. Figure 5 plots the intra-subnet handshake RTTs below one millisecond. Clearly, the values exhibit sharp quantization around the multiples of 125 microseconds. We observed the same phenomenon in other `LBL1` RTT measurements, too—inter-subnet handshake RTTs and RTTs measured between TCP data packets and their ACKs (we refer to these further as data RTTs). In `LBL1`, 96% of all the intra-subnet and 87% of all the inter-subnet handshake RTTs lie in the sub-millisecond range, which elevates the importance of understanding the observed RTT quantization phenomenon. And more generally, we want to illustrate here the sort of methodical thinking that goes into calibration, along with the crucial question of whether somehow this was a real networking phenomenon, or just a measurement artifact.

As 125 microseconds corresponds to 8 KHz, we might naturally hypothesize that some device or software related to our measurement contains an oscillator or a sampling process operating at this frequency. Each of the following components might use an 8 KHz clock: end-hosts, production network equipment (switches or routers) or our capturing apparatus. We might view it as unlikely that such a relatively course timer would occur within the networking component of an OS kernel, or NICs or production switches, since doing so would introduce significant unnecessary delay. However, if we find that the quantization reflects a genuine networking effect, then we will have uncovered an area for significant performance improvement.

In a 100 Mb/s Ethernet network, sending a full-sized packet of 1,500 bytes takes 120 usec, very close to the multiples of quantization times we observe. In an attempt to locate the source of RTT clustering, we extracted all full-size packets from a single `LBL1` trace and calculated the inter-arrival times between those packets. We found two spikes in the plot: a bigger one around 120 usec and a smaller one at a few usec. The presence of the latter suggests that the `LBL1` traces contain impossible timings, presumably corrupted by our capturing apparatus. Looking at the RTTs of the

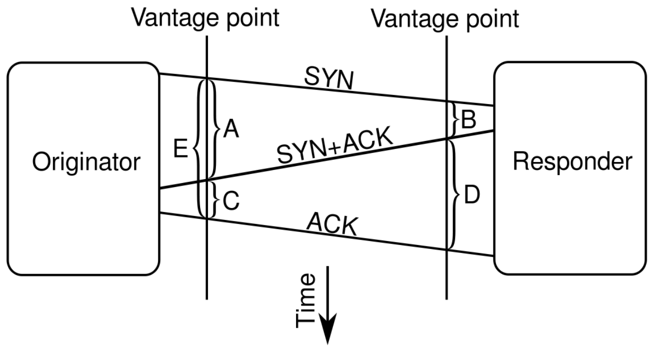

Fig. 6. A/B/C/D/E components of a TCP handshake.

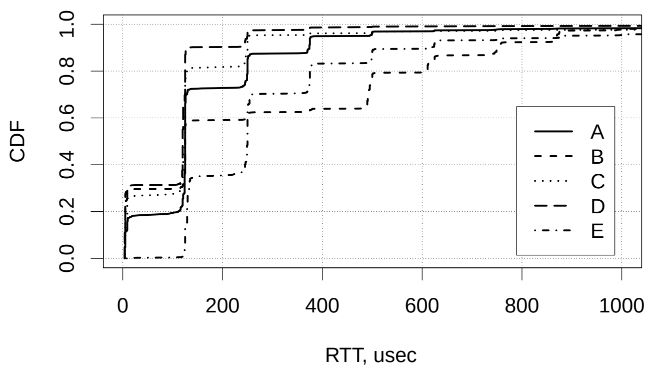

Fig. 7. `LBL1` sub-millisecond intra-subnet handshake RTT components.

full-sized data packets in inter-subnet connections also reveals impossibly small values. The process causing RTT corruption may simply be the queueing in one of the capturing buffers. If upon the data packet arrival the buffer is full, the corresponding ACK packet may have enough time to arrive right behind the data packet and appear to come immediately after it. Another source of error concerns the capturing apparatus location with respect to the sender and receiver. In the `LBL1` capturing setup, the mirroring switch operates using store-and-forward, not cut-through, meaning that it waits to receive all of the bytes in a packet and only then forwards it. Thus, a full-sized data packet takes at least 240 usec to arrive at the tracing machine (120 usec due to each of transmission by the sender and by the mirroring switch). In the case of intra-subnet communication and the production switch operating cut-through, the above scenario can readily lead to ACK packets arriving at the monitoring switch while the corresponding data packet still remains in the monitoring switch's queue, causing the two packets to follow one another back-to-back.

To finally untangle the RTT quantization phenomenon, we again turned to the handshake RTTs that, in the light of the above findings, have several important properties: (i) small packet sizes, which diminishes the differences between store-and-forward vs. cut-through switch architectures, as well as the contribution of packet transmission time; (ii) lower likelihood of encountering competing load due to the host's own prior activity, which could impair RTT fidelity; and (iii) the opportunity that three-way handshakes provide to assess RTTs both at the sender side (SYN/SYN+ACK) and the receiver side (SYN+ACK/ACK).

We proceed by separating each handshake RTT into five components, which we refer to as A/B/C/D/E. A and B reflect SYN/SYN+ACK time differences as seen from a vantage point near the connection originator and responder, respectively. C and D reflect SYN+ACK/ACK time differences, again from a vantage point near the connection originator and responder. Finally, E corresponds to the full SYN/.../ACK time difference (i.e., between the final packet of the handshake and the initial SYN) as seen from a vantage point near the connection originator. To exclude any router delays, we limited this analysis to intra-subnet connections. The list of monitored hosts we devised for `LBL1` in [11] allowed us to determine measurement vantage points: if the source IP address in a SYN belongs to a monitored host, then we know we have a vantage point for that connection close to the originator. Conversely, a destination IP address in the SYN belonging to a monitored host indicates we have a vantage point near the responder.

A and D components contain the path RTT between a vantage point and a host as well as the host's processing delay; B and C reflect only the end-host's processing delay; and E provides the true handshake RTT, factoring out the vantage point position. Figure 7 shows cumulative distribution functions for the five components, including several striking phenomenon. Firstly, we observe quantization for the B and C curves, meaning that it arises either from processing at the end hosts, or we are indeed dealing with a measurement artifact. Secondly, we find a considerable share of the A and D RTTs exhibiting values as small as several microseconds, meaning that responders indeed can generate SYN+ACKs (and, for originators, ACKs) essentially instantaneously. *However* we do not observe any E RTTs in that time range, suggesting that for those connections either A or C must necessarily cluster at a multiple of 125 usec.

We view the most robust explanation for the quantization phenomenon as it reflects a measurement artifact, as follows. Per our earlier description, in a handshake the A and C components comprise two non-intersecting

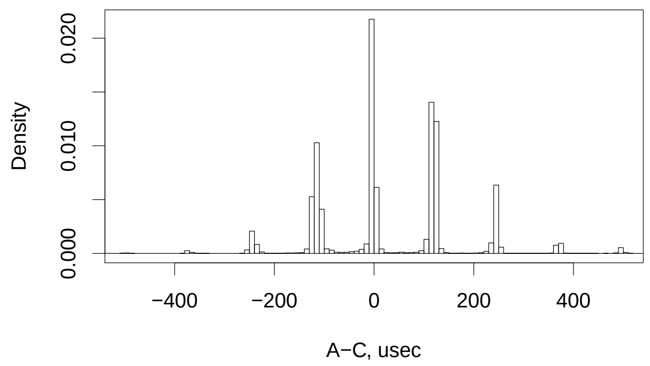

Fig. 8. A minus C for each `LBL1` handshake.

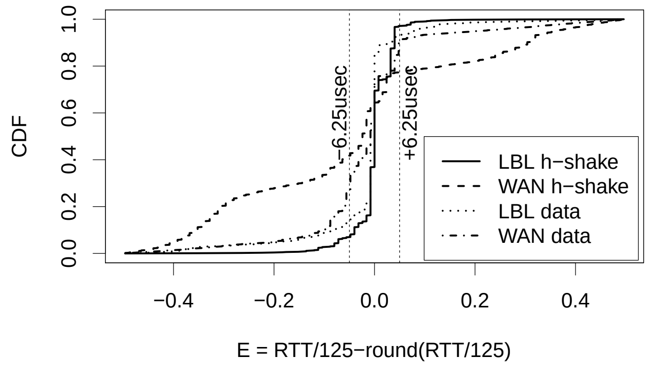

Fig. 9. `LBL1` RTT precision.

parts of the path. C will not include any quantization due to the production switch, since it only reflects end-host processing latency (and any potential measurement artifacts). Given the diversity of end host hardware vendors and operating systems in LBL, we have difficulty envisioning how they could *synchronously* produce the quantization pattern we observe. According to Figure 8 quantization occurs with equal probability in both A and C, and since the only part of the infrastructure these two components share is our measurement apparatus, we unfortunately must conclude that the observed quantization anomaly arises from our `LBL1` apparatus, rather than any actual networking phenomenon.

Earlier we have hypothesized that the observed quantization may be due to an 8 KHz timer. Now we aim to explore this possibility by plotting the offset of each observed RTT from the closest multiple of 125 usec. If we find all such offsets to turn out small, we can treat the RTTs as having 125 usec precision imposed by the capturing apparatus, even regardless of knowing the true source of the quantization. Figure 9 shows the cumulative distributions of $E = RTT/125 - round(RTT/125)$ for four types of traffic: TCP handshakes for connections staying inside LBL or crossing the enterprise's boundary (WAN), and TCP data packet RTTs for the same two localities. For handshakes we used the total RTT, and for data packets we calculated the time difference between a data packet and the corresponding ACK regardless of vantage point's position, which means that those data packet RTTs that we calculated close the responder contain mainly the end-host processing delay. We nevertheless decided to include these timings to see if their precision differs from the full path RTT observed at a vantage point close to the connection originator. The plot demonstrates that almost all RTTs of packets confined to the enterprise lie close to multiples of 125 usec. To a smaller degree we also find this for the WAN data packets. On the contrary, WAN handshake RTTs exhibit very high offsets.

Having 8 KHz precision in the capturing apparatus would mean that all observed timings would be close to multiples of 125 usec. But in Figure 9 we can clearly see that this is not the case. Therefore we cannot treat the `LBL1` timings as having the 125 usec precision. Since the biggest deviation from a multiple of 125 usec occurs in WAN data traffic, which tends to have bigger RTTs, the plot hints that an 8 KHz timer may manifest only for small RTTs. Indeed, we found an anti-correlation between the RTT value and its offset for all traffic types. This does not however explain why WAN data packets cluster close to the multiples of 125 usec even though WAN handshakes do not—one would expect the two classes to have equally high RTTs. We find it possible that an 8 KHz timer fires only under high load. TCP dynamics often leads to trains of multiple packets, while handshake packets are free to arrive during periods of silence. In addition, the two factors influencing the work of the timer may not be independent, since small RTTs may cause higher instantaneous load.

Fortunately, the `LBL2` handshake RTTs do not exhibit signs of quantization (see Figure 10). The plot also shows fairly good agreement between the distributions of intra- and inter-subnet delays for the two sets of traces.

Besides handshake RTTs, we also explored the RTTs of TCP data packets, i.e., the time difference between data packets and their ACKs. We took care to consider only data packets coming for monitored hosts, thus ensuring proximity of the vantage point to the sender. Figure 11 shows the distributions of the data packet RTTs. We find the `LBL2` RTTs consistently smaller than

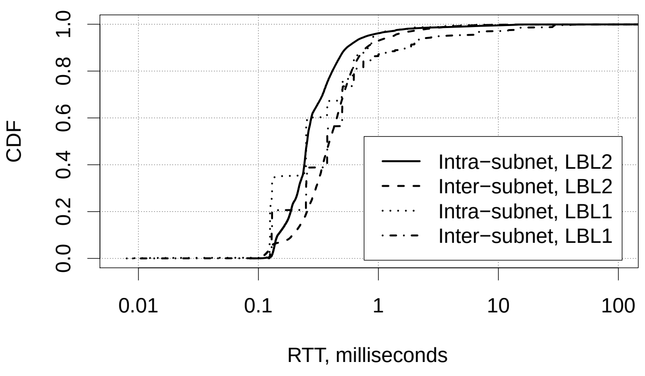

Fig. 10. LBL1 and LBL2 handshake RTTs. Note, for clarity we truncate the plot by omitting extreme values to the right.

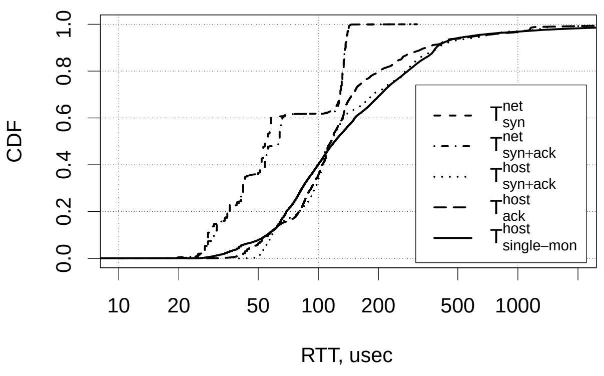

Fig. 12. LBL2 handshake network and host delay. We truncate the plot by omitting extreme values to the right.

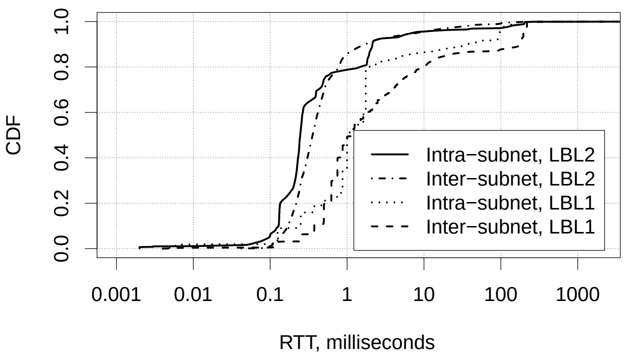

Fig. 11. LBL1 and LBL2 data RTTs. Again for clarity we truncate the plot by omitting extreme values to the right.

for LBL1, which makes sense since the newer traces include 1 Gbps links not present in the older traces.

While exploring the LBL2 handshake and data RTTs, we found a number of TCP connections with both the sender and the receiver monitored. 11,588 such double-monitored connections appear across 72% of LBL2 traces. The existence of these connections opens an intriguing possibility: we can fully separate the network delay from the end-host processing delay. Consider a host establishing a TCP connection, for which the initial SYN packet appears at the vantage point close to the sender at time t^{orig}_{syn}. At time t^{dest}_{syn} the SYN packet will pass the vantage point close to the monitored destination host, thus making $T^{net}_{syn} = t^{dest}_{syn} - t^{orig}_{syn}$ the time the SYN packet spent in the network. The SYN+ACK packet observed at the destination vantage point allows us to calculate the time it took the destination host to generate the packet: $T^{host}_{syn+ack} = t^{dest}_{syn+ack} - t^{dest}_{syn}$. Similarly, we consider $T^{net}_{syn+ack} = t^{orig}_{syn+ack} - t^{dest}_{syn+ack}$ to be the time the SYN+ACK spends in the network and $T^{host}_{ack} = t^{orig}_{ack} - t^{orig}_{syn+ack}$ the time it took the originator to generate an ACK in response to the SYN+ACK. We match the respective connections as seen at the originator and responder by the five-tuple.

The results in Figure 12 show good consistency between T^{net}_{syn} and $T^{net}_{syn+ack}$ as well as between $T^{host}_{syn+ack}$ and T^{host}_{ack}. Surprisingly, the plot indicates that a packet spends *less* time in the network than it takes an end-host to generate a response packet, meaning that processing and not network delay dominates the LBL2 handshake RTTs. To corroborate our observation of the high host processing delay, we also plotted the handshake host delay ($T^{host}_{single-mon}$) of the connections for which we monitored only one of the two communicating machines.

Even though RTTs in the LBL2 traces do not show signs of quantization, and thus appear more reliable than those in the LBL1 traces, we still cannot fully trust them. From Section IV we know that capturing the two traffic directions separately led to perturbations in packet timestamps. We “flipped” 6.9 million TCP packets in order to restore the correct causal order. This implies that undoubtedly packets that appeared in the correct causal order, but with compressed or stretched RTT values, also exist. Unfortunately, we know of no apparent way to detect such packets and repair their timings. We can, however, conclude that the errors are generally small: we found that most flipping reflects sub-millisecond timescales, per Figure 13, which shows that 98.9% of intervals lied below one millisecond.

To summarize our experience in calibrating timing fidelity, the switch-based measurement is ill-suited for fine-grained RTT and delay measurements. As we found,

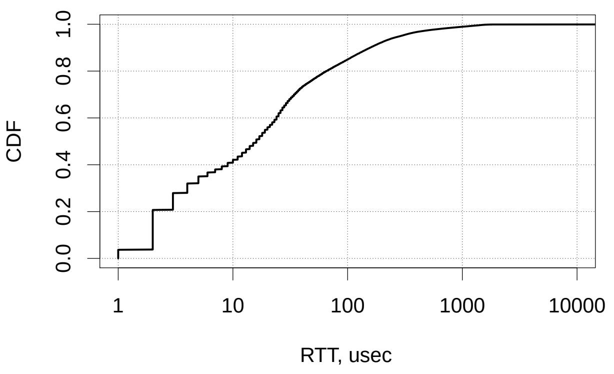

Fig. 13. LBL2 absolute flipping time difference.

neither in LBL1 nor in LBL2 can we trust RTTs smaller than one millisecond. Post-hoc techniques for fine-grained (order of microsecond) latency measurements described in literature either focus on calculating aggregated latency [7], [9] or use interpolation to improve accuracy [8]. We believe that a better way to approach the problem is to use specialized hardware [4].

VIII. Phantom suppression

After going through all the calibration stages described in the previous sections, we continued to the analysis phase of our study. As our first task we chose to calculate protocol prevalence figures inside subnets. The results baffled us: we observed an abnormally high number of ARP request packets. The explanation turned out to be that simultaneously recording packets from multiple ports on the same switch causes broadcasted packets to appear multiple times in the traces. In [11] we called such replicated packets *phantoms*. Since their presence may unduly skew the traffic mix proportions, we developed a methodology for identifying redundant copies of a packet so we can then remove ("suppress") the extraneous ones.

Phantoms are a good example of a phenomenon that only pops out when one starts to use the data and is not well predicted during the first stage of calibration by solely focusing on what was recorded. This once again demonstrates how unpredictable the calibration process may be, and highlights the need for iterations during calibration.

In the absence of any truly robust way of identifying phantoms in the LBL1 traces, we developed an empirical rule based on interarrival time between the identical packets. By examining the interarrival intervals, we aimed to establish a threshold for removing the duplicates. We needed to determine a value small enough to avoid the risk of removing the packets that represented true networking events rather than the copies made by the switch, but large enough to ensure we would not miss any phantoms. To identify an apt threshold, in [11] we found it highly helpful to consider *sole-sourced* packets, i.e., packets that we could with confidence state that the sending host sent exactly one instance of the packet in a given trace. (Note, that packet could still have duplicates produced by the switch. See below for discussion.) Analyzing the interarrival times between the copies of such packets let us then find a plausible suppression threshold and estimate the number of false positives—the packets erroneously labeled as duplicates that in fact constituted true networking events.

Thanks to our use of a different recording methodology, the LBL2 traces contained information that allowed us to refine this methodology and assess the accuracy of the findings made in [11]. While the older traces offered no way to determine which switch port recorded a given packet, the use of VLAN tags in the newer traces provided such a mapping. This mapping enabled us to assess the accuracy of the previous evaluation in [11] based on identifying sole-sourced packets. In that work, we defined an Ethernet broadcast packet as sole-sourced if (i) the trace exhibited 5 or fewer copies of the packet, and (ii) the intervals between adjacent copies spanned less than 100 msec. We restricted the number of copies to 5 to reflect the maximum number of ports simultaneously monitored in the older traces, and chose 100 msec as a plausible interval threshold because, first, we find it hard to envision a switch process introducing larger delays between replicated packets, and second, the distribution of packet interarrival times showed that they tended to be either much smaller or much larger than 100 msec.

As we note in [11], the above two criteria do not provide ironclad identification of sole-sourced packets—in some cases we may indeed mis-classify closely spaced identical transmissions from the same host as sole-sourced. For the LBL2 dataset, we can make the rule more robust by adding a third criterion: (iii) all of the copies originate from distinct switch ports. Using the VLAN tags we can calculate the number of cases where applying only the first two conditions failed to identify a truly sole-sourced packet. After changing the number of copies from 5 to 10 to match the number of monitored ports in the newer traces, we ran the check and found *no* cases where we needed the third rule to distinguish between a true and a false sole-sourced packet. Thus, we confirm that the rules (i) and (ii) suffice for identification of sole-sourced packets.

Next, we examine the intervals between the extracted sole-sourced packets. We compare the results with the ones for LBL1 by giving the latter in brackets after the LBL2 figures. We found 104.3M (20.4M) intervals between sole-sourced packets. The distributions of intervals for the older and newer traces turned out to be fairly consistent: in 74% (60%) of traces the interval never exceeded 1 msec, the 99th percentile across all traces never exceeded 0.2 msec (2 msec). 99.998% (99.998%) of the intervals lie below 5 msec; all of the intervals lie below 16.6 msec (58 msec). These figures allow us to come to the same conclusion as in [11]—the 5 msec threshold correctly identifies nearly all duplicates.

We used a nearly simplest possible scheme for eliminating duplicates in the LBL1 traces. We considered every packet with a hash value already seen within 5 msec in the past as a phantom. This approach may seem too simplistic, and indeed it allows for unnecessary suppression. For instance, in the case when a host sends two identical packets shortly one after another, there may be 10 hashes each within 5 msec from the next one, and consequently our scheme will delete 9 packets, leaving only the first one. We decided not to include a simple check for the number of packets being suppressed (must be 5 for LBL1 or 10 for LBL2) for two reasons. First, without VLAN tags we have no way of knowing exactly how many of the duplicates were in fact real packets. Even though the maximum number of possible duplicates was 5 for the LBL1 traces, on many occasions we observed fewer duplicates, for example because one or more switch ports were inactive. Secondly, we did not in fact need such a rule. After we found that the 5 msec interval covers most sole-sourced duplicates, we set out to assess the number of *false positives*—the cases where the above threshold marks true events as duplicates. To spot false positives we switched from the sole-sourced packets to all broadcast packets and counted how many times the number of duplicates within the 5 msec suppression threshold exceeds 5 for the older and 10 for the newer traces. Since we know that our capturing apparatuses allowed no more than this many copies of a packet, such events necessarily indicate that we would suppress several true (though identical) packets. In the LBL1 traces, the simplistic algorithm based only on the interval between packets yielded 150 false positives out of 7.8M unique broadcast packet payloads. The minuscule false positive rate rendered more sophisticated suppression techniques unnecessary.

To find if we could use the same simple technique for the LBL2 traces, we repeated the false-positives check. 12.1M unique broadcast packet payloads turned out to produce 158K false positives. The sharp contrast between the factor of 1,000 increase in the number of false positives with the factor of only a 1.55 increase in the number of unique payloads indicates that this technique would cause much unwanted over-suppression of true packets. To avoid this problem, we need more refined suppression techniques for the newer traces. We assessed three alternatives, which we discuss here in ascending order of the amount of information needed for accurate suppression:

Algorithm 1. Time based only. The only suppression rule: remove the packet if there was a packet with the same hash within the past 5 msec. We used this simple approach in [11].

Algorithm 2. Packet count based. In addition to the rule of Algorithm 1, count how many packets with this hash we have suppressed so far. If the number of packets goes beyond the maximum possible number of switch replications, we must have included a new true packet, so we cease suppressing duplicates at that point and start the count over. We can apply this approach to both the LBL1 and the LBL2 traces.

Algorithm 3. VLAN number based. In addition to the rule of Algorithm 1, we track the VLAN IDs of the packets with the same hash suppressed so far. Since all duplicates of a single packet have different VLAN IDs, hitting the same VLAN ID twice unambiguously indicates a new packet. This approach, however, has a caveat. If a host sends two real, identical packets close to each other in time, their duplicates may overlap. In this scenario the algorithm will under-suppress.

We evaluated these three algorithms on the LBL2 traces and divided the results into two categories. The first with less than 10 duplicates and no repeating VLAN IDs among them. Such cases will yield the same results for all the three algorithms. The second category includes the over-suppression cases when the algorithms will leave different number of packets. Considering only the second category, the number of packets left after suppression for Algorithm 1 was 525K; for Algorithm 2, 968K; and for the Algorithm 3, 1,762K. The mismatch between the Algorithm 1 and the Algorithm 3 reflects more than a factor of three, confirming the hypothesis that the naïve algorithm over-suppresses a great deal in the LBL2 traces. (The dominant group in the over-suppressed packets consists of identical IPv4 multicast packets appearing close to each other.)

The difference between the results produced by the three algorithms appears less drastic if we compare the number of over-suppression cases to the total number of cases requiring suppression. The second category has the same number of packets as left after running Algorithm 1 (525K), since it constitutes the most aggressive approach. In total, 57M cases required suppression, meaning that in more than 99% of cases, Algorithm 1 suffices. Thus, in conclusion, we find that the suppression technique used in [11] has sufficient accuracy to suffice in the vast majority of cases; using more sophisticated approaches would not have gained much additional fidelity.

IX. Conclusion

We aim in this work to present a general approach to the problem of how to thoroughly calibrate empirical data in order to identify, and if possible remedy, shortcomings and biases present in the data due to the process by which we collected it.

In general, calibration proceeds in a *progressive* fashion, beginning with the most basic properties of the data and, with those understood, tackling each of the next most basic in turn. This procedure often has an *iterative* nature, where the investigation of more complex properties can require revisiting earlier calibration steps in order to further refine them. To the extent possible, we also advocate for performing a degree of basic monitoring during the measurement process itself, as a way of detecting systematic problems in time to perhaps correct them.

We illustrate our calibration methodology using a case study of network trace measurements captured from switches within an enterprise. We showed how addressing one of the most basic properties of the data—how many packets did the apparatus fail to record?—actually required delving into issues regarding packet ordering and measurement timestamping. After untangling these considerations and partially remedying some deficiencies regarding packet ordering, we examined later calibration stages regarding assessing the topology of the enterprise systems present in the measurement, further examination of the fidelity of the timing information, and identification and removal of measurement duplicates.

Some of our calibration steps followed algorithms and techniques we developed previously in [11], where we described calibration of a similar set of traces, albeit collected using a differently designed capturing apparatus. The differences in the capturing process between the two sets of traces allowed us to assess the accuracy of some of the calibration techniques developed in the earlier work; in general, we conclude that the techniques provide a good degree of accuracy and provide reliable means for calibrating enterprise switch measurements.

While such calibration efforts often prove labor-intensive, they arguably play a vital role in establishing the ultimate soundness of any subsequent analysis based on the data.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM'08, pages 63–74, 2008.

[2] J. Bennett, C. Partridge, and N. Shectman. Packet Reordering is Not Pathological Network Behavior. *IEEE/ACM Transactions on Networking*, Dec. 1999.

[3] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *ACM Internet Measurement Conference*, IMC'10, pages 267–280, Nov. 2010.

[4] D. A. Freedman, T. Marian, J. H. Lee, K. Birman, H. Weatherspoon, and C. Xu. Exact temporal characterization of 10 Gbps optical wide-area network. In *ACM Internet Measurement Conference*, IMC'10, pages 342–355, 2010.

[5] IEEE Standards Association. IEEE 802.1Q Virtual Bridged Local Area Networks. http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf, 2005.

[6] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *SIGCOMM Comput. Commun. Rev.*, 32:75–88, July 2002.

[7] R. R. Kompella, K. Levchenko, A. C. Snoeren, and G. Varghese. Every microsecond counts: tracking fine-grain latencies with a lossy difference aggregator. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM'09, pages 255–266, 2009.

[8] M. Lee, N. Duffield, and R. R. Kompella. Not all microseconds are equal: fine-grained per-flow measurements with reference latency interpolation. In *Proceedings of the ACM SIGCOMM 2010 conference on Data communication*, SIGCOMM'10, pages 27–38, 2010.

[9] M. Lee, S. Goldberg, R. R. Kompella, and G. Varghese. Fine-grained latency and loss measurements in the presence of reordering. In *ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, 2011.

[10] B. Nechaev, M. Allman, V. Paxson, and A. Gurtov. A preliminary analysis of TCP performance in an enterprise network. In *INM/WREN*, April 2010.

[11] B. Nechaev, V. Paxson, M. Allman, and A. Gurtov. On Calibrating Enterprise Switch Measurements. In *ACM Internet Measurement Conference*, IMC'09, pages 143–155, Nov. 2009.

[12] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. ACM SIGCOMM '98, pages 303–314, 1998.

[13] V. Paxson. End-to-End Internet Packet Dynamics. In *ACM SIGCOMM*, Sept. 1997.

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

[15] V. Paxson. Strategies for sound Internet measurement. In *ACM Internet Measurement Conference*, IMC'04, pages 263–271, Oct. 2004.

[16] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath TCP. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM'11, pages 266–277, 2011.

[17] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM'11, pages 50–61, 2011.