

HostView: Annotating end-host performance measurements with user feedback

Diana Joumlatt, Renata Teixeira
CNRS and UPMC Sorbonne Universités

Jaideep Chandrashekar, Nina Taft
Intel Labs Berkeley

1. INTRODUCTION

Network disruptions can adversely impact a users' web browsing, cause video/audio interruptions, or render web sites and services unreachable. Such problems are frustrating to Internet users, who are oblivious to the underlying problems, but completely exposed to the service degradations. Ideally users' end systems would have diagnostic tools that can automatically detect, diagnose and possibly repair, performance degradations. Hopefully, this can be done without user intervention. Clearly, the first step for any such (end-host) diagnostic tool is a methodology to automatically detect performance degradations in the network that can affect a user's perception of application performance.

While empirical network performance studies have been carried out in the past, the bulk of these have focused on well established performance metrics—delay, loss rate, throughput. Very little attention has been devoted to how these metrics affect end-users' interactions with applications that rely on the network. Understanding the more subjective quality of experience has been limited to a few niche applications, for instance VoIP [1], online-gaming [2], and video playback [3, 4]. Building such application specific models are useful in improving the specific application, but cannot be easily generalized to capture a particular user's "online experience", which is likely to span a number of different applications. Moreover, building application-specific models requires a great deal of domain knowledge.

We argue for building a general purpose tool that can detect network performance degradations that affect users' quality of experience. Moreover, we wish to carry out this detection with the user's mix of applications and working environment. This tool is fundamentally hard to build. First, perception of performance varies dramatically across users and even for the same user according to her mood and expectations. A user is more likely to expect better performance from her network at work, than from a free wireless at a coffee shop. Second, the quality of experience is very tightly coupled to the application. For instance, a sudden increase in round-trip times (RTTs) may have no discernable impact on a video connection (playback buffers mask jitter), but it can make a remote login session unusable. These factors clearly imply that we cannot really study network performance (or its degradation) without taking into account the application, the end-user, and the environment. Understanding how to construct a diagnostic tool requires analysis of network performance data annotated with users' perception of network quality. However, no such dataset exists today! A few end-host measurement datasets exist [5–7],

but these lack the all important annotations of users' perceived quality. Other studies incorporate user feedback with the performance measurements, for specific applications [4], but the data is not collected at the end-host. Clearly if one is to correlate a user's perception of performance with that of the network, applications and mobile environment her device operates in, the data collection utility must reside on the end-host itself.

In this paper, we describe the design of HostView, an end-host measuring tool to collect network performance data annotated with users' perceived quality of the network. The questions of *what* data to collect, and *how* to collect it are fundamental design questions that are also driven by the user. Certainly, the more data a tool collects the better it will be able to do diagnosis. But privacy concerns, while subjective, will govern the types of data collected. The mechanism for collecting data is also critical because heavy handed methods that affect the basic performance of the machine will deter a user from installing such a tool. Thus a study of the overhead of any measurement method is imperative in the design of end-host tracing tools. We describe how we came to our design decisions regarding what type of data to collect, and using our first prototype we present the overhead of candidate collection techniques (§2).

Another key design question focuses on how to extract feedback from users across different network conditions with a minimum level of annoyance (§3). We build a second prototype of HostView¹ that includes an algorithm to sample user experience three times a day. Our two-week experiments with a small set of users show that users' unpredictable behavior is an important challenge for any sampling algorithm. Our long-term goal is to collect traces via a large-scale campaign that will enable a broad research agenda.

2. END-HOST DATA

We concentrate on collecting three kinds of data: (i) traffic and network performance statistics, (ii) application level context (for traffic flows), and (iii) system performance and environmental data (network type, etc). This section explores what specific data should be collected within these broad categories and also evaluates the accuracy versus overhead tradeoff of candidate collection methods. The performance study was carried out with a first implementation of HostView that runs on MacOS X and Linux, via a small pilot with 7 students from LIP6 who ran the tool on their laptops for 2 weeks.

¹<http://cmon.lip6.fr/EMD/>

2.1 Network performance

Active network probing, as a mechanism to measure network performance, is appealing because it sidesteps issues related to recording potentially private information (as compared to passively recording traffic). However, the performance inferred by active probing might not reflect that of any particular application: the destinations are different; probe packets may be handled differently from those of applications; the probing may happen outside of performance episodes, and so on! Thus, to overcome these difficulties, HostView uses passive measurements and incorporates a number of mechanisms to allay privacy concerns with recording traffic. With the packet traces, we can infer RTTs, losses, and reachability issues of active application connections [8]. We log the traffic, rather than simply log summary statistics as in [7], because the summaries are not fine-grained enough to allow detailed analysis on specific connections. Logging all the traffic packets is easily accomplished with the `libpcap` library; which has been extensively tested and optimized. Previous results show that the overhead with `libpcap` is under 20% in the worst case [9] (full packet capture on 100Mbps Ethernet). In our own testing, we never saw the overhead exceed 5%; thus, we do not expect the overhead of `libpcap` to be significant.

Recording packets raises issues with some users about privacy; few people are readily willing to run such a measurement tool. To understand this issue better, we conducted a user survey of 400 computer scientists to assess user discomfort with specific kinds of data being logged. Our results, summarized in [10], lead us to believe that a tool that is restricted to recording packet headers (and not the entire packet) and which appropriately anonymizes the traces, would be acceptable to a large fraction of users.

To this end, HostView incorporates a number of privacy protection features. First, it removes any information that identifies the host: users are identified by a randomly generated id, which is used to prefix trace file names. Source IP addresses are anonymized with a SHA-256 hash. In addition, the traces are periodically uploaded using `s-ftp` and the server where the traces reside has very restricted access controls. Third, HostView allows users to *pause* the logging (all logging is disabled in this period) in one hour increments, when the user carries out any activity they do not want recorded.

2.2 Application context

To understand how individual applications are affected by network events, we also need to associate network flows (reconstructed from the packet traces collected by `libpcap`) with the application (executable) that is terminating the flow and also correctly identify the application *protocol* embedded in the stream of packets. The traffic stream seen at the end-host is a multiplexing of the traffic from several individual applications (Firefox, Yahoo Messenger, Skype, etc.), each of which uses a multiplicity of application protocols (HTTP, FTP, SMTP, RPC, etc.). Using destination ports alone (without other information) of a network flow is reputed to have limited value in understanding which application generated the flow. This is because the destination port numbers are often overloaded, or used in unorthodox ways (applications often tunnel their protocols over HTTP to traverse firewalls, as an example). In the following, we describe mechanisms to associate these two labels with net-

work flows seen at the end-host.

Identifying Application name: There are several APIs available that allow mapping network flows to application names (in short, *app name*) or process identifiers—ETW [11] and `lsnf` are two in particular. However, they are often platform specific (ETW is only supported in Windows), or come with very high overhead (`lsnf`), which makes them hard to use for a broad deployment. Among the options we considered, the *gt* toolkit [12] was found to have the best balance of platform support (it runs on our target platforms), and low overhead, and this has been integrated into HostView. This particular method periodically samples the kernel socket tables, which contain mappings between applications and sockets, and associates outgoing flows with the appropriate socket entry. This method has an intrinsic tradeoff between sampling interval and coverage: shorter intervals are likely to label more flows, but at a higher cost. Longer sampling intervals reduce the overhead, but tend to label fewer flows (missing those that start and terminate between sampling periods). This tool has been profiled extensively in previous work [12]. The results showed CPU loads of 5% with 1 s polling intervals, and 20% when the interval is reduced to 125 ms. In our own testing, we also ran it with a more extreme sampling interval of 0.01 s and with this sampling rate, we observed average CPU loads of 59%. While this did give us a very small increase in classification coverage (a few percentage points), the substantial increase in the CPU loads makes this operating point undesirable; the default setting of 1 s seems like a reasonable choice, and this is the sampling interval that is used in HostView.

Identifying Application protocol: The naive method to associate a network flow with the application protocol is to simply consult the IANA list of well known port numbers (`/etc/services` in most *nix distributions). This method tends to be unreliable when applications use non-standard or dynamic port numbers. More robust application protocol classifiers include those that scan for known signatures in payloads [13], and statistical methods that apply learned traffic models to traffic distributions. While such methods are well matched for traffic collected at network gateways [14], they have costly CPU requirements, and a careful evaluation of their feasibility on end-hosts has not yet been carried out.

To understand whether the more complex classifiers can practically be deployed on end-hosts, a first implementation of HostView incorporated the *l7* classifier [13]. This particular instance has built into it a comprehensive database of protocol signatures. We choose this particular payload classifier since it is considered the authoritative payload-based classifier and has been extensively used to determine “ground-truth” in other studies of classification accuracy. We do not include port-based classification or any of the statistical-based methods in this version of the tool, because these can be run offline using the packet header traces we already export to the server (*l7* inspects the full packet payload, and hence cannot be run on packet headers alone). In the data collected from our pilot deployment over 7 users, we observed that CPU load of *l7* is directly proportional to traffic load; we also observed frequent episodes where the CPU load spiked very high (this was found to correlate with bursts in traffic volume). Since we expect traffic to be bursty, it would seem that running *l7* on a user’s production end-host

Output	Method	TCP		UDP		Total traffic	
		Flows	Bytes	Flows	Bytes	Flows	Bytes
Protocol	<i>l7</i>	99.20%	99.90%	47.72%	13.66%	60.08%	90.04%
	port	99.60%	99.85%	87.82%	69.90%	90.65%	96.39%
	<i>l7=port</i>	94.77%	95.84%	46.76%	13.51%	58.29%	86.43%
				93.69%	98.41%	94.89%	95.82%
App Name	<i>gt</i>	99.56%	99.89%	10.30%	1.36%	31.73%	88.63%
				79.13%	96.13%	83.87%	99.64%

Table 1: Application classification coverage

would not be recommended and we will have to resort to alternate methods of performing the classification.

Using the traces from our initial pilot, Table 1 presents the fraction of flows and bytes, that were correctly classified either by the *l7* method, and the port inspection method. The row labeled *l7=port* shows the flows and bytes for which both methods gave the same classification. The last row presents the fraction of flows and bytes with which *gt* could associate an application name.

Comparing the results of port- and payload-based classification in Table 1, we see that the two techniques return the same application protocol for 86.43% of bytes. This fraction is even higher if we only consider TCP connections (over 94% of flows and bytes), which confirms a recent study with residential DSL customers [15]. We find surprisingly that the port-based classifier does reasonably well. Similarly, *gt* is able to assign an application name to almost all the TCP connections. Therefore, *the combination of port inspection to obtain application protocol and gt for application name suffices for labeling TCP connections*. Although a different application mix would lead to different classification accuracy for port inspection, the application name should allow us to identify these cases. For instance, one of our traces contains BitTorrent traffic. Although the connections use port 80, the application name contains the string “BitTorrent”.

The fraction of flows and bytes labeled by *l7* and by *gt* is much lower for UDP. We examine UDP destination ports of all unlabeled flows both for *gt* and *l7* and find that the vast majority of these flows are broadcast and multicast traffic on the local network (for instance, NetBIOS, MDNS, and UPnP). If we assume that these ports are always used consistently, and hard code these into the classifier, the results are remarkably better (over 90% accurate classification). *Port inspection is the only technique to infer the application context for UDP flows*.

HTTP traffic accounts for a very large fraction of the bytes seen in our traces. Given our overall goal, we feel it advantageous to also record the content-type fields (video, audio, images, or text) to gain an understanding of the type of application being transported. However, this does require some form of online payload inspection, just to extract the leading portion of the packet (we find that the first 500 bytes are sufficient to identify the content-type in most cases).

To sum, HostView logs application names associated with open sockets every second using the *gt* toolkit. It does not incorporate any online methods to identify the application protocol; this identification is relegated to the back end server where the traces are uploaded. Finally, HostView tries to identify the content-type from looking at the first few packets of potential HTTP responses.

2.3 System performance and environment

We also log system and environment data to broaden our understanding of the end-host’s performance. 1) HostView logs *CPU load* every second and this data is uploaded to a back-end server. 2) The *active network interface* is recorded whenever it changes; if the active network is wireless, a pop-up asks the user to assign a label to the network (work, airport, coffee-shop, etc) when the wireless network SSID is encountered for the first time. As a privacy mechanism, HostView only logs a hash of the SSID (and not the SSID text string) along with the user supplied descriptive label for the network. HostView periodically logs a few wireless network performance indicators 3) *received signal strength*, 4) *noise level* and 5) *last transmission rate*. 6) HostView also attempts to find the home *autonomous system (AS)* that the end-host is in each time its IP address changes. This is carried out with a 3-hop limited traceroute to google.com after which the IP addresses of each hop are mapped to ASes using an IP-to-AS mapping tool; only the AS numbers are actually exported from the end-host. We collect the AS information to have the ability to understand how a user’s perception of performance can vary when connected to different ASes.

3. USER FEEDBACK

While network performance can be objectively measured, the actual impact felt by the user is much harder to capture and quantify. At the same time, capturing the user’s perception of network performance is crucial to understanding how, and to what extent, network disruptions affect different users and applications. Motivated by the extensive user studies research carried out in the HCI community, we incorporated user feedback mechanisms directly into our tool.

3.1 Questionnaire design

We capture user feedback by asking people to fill out a set of questions about their network connection quality. Interesting tradeoffs arise when designing such a questionnaire; on the one hand we want the questionnaire to be short so that users will take the time to fill it out, but on the other hand the more information we obtain the better for our research. We limited our questionnaire to 5 questions (a common amount in HCI surveys [16]) because more people are willing to participate when the questionnaire takes less than a minute or two to complete. User feedback can be both quantitative (e.g. rating performance on a scale, or things for which statistical summaries can be produced) and qualitative (e.g., free-form text). Multiple choices are easier to interpret than open format questions, but they can limit the feedback a user can provide. To balance these issues, our questionnaire uses a mix of both quantitative and qualitative. For example, we ask the user to rate the quality of their perceived network connection on a standard Likert scale [17] from 1 to 5 (widely used scale in survey research). We also ask the user to list which applications (if any) experienced problems accessing the Internet in the last 5 minutes prior to the questionnaire. For qualitative input, users can enter (via a free form text box) any additional comments they would like describing their computer’s performance (at the application/network/machine level); this includes even potential hypotheses the user might have as to why poor performance occurred.

3.2 Triggering the questionnaire

There are two approaches to trigger the questionnaire in order to elicit user feedback: system- or user-driven. The first approach, Experience sampling (ESM) [18], is widely adopted in the HCI community [16]. By triggering the questionnaire using system measurements such as throughput or CPU load, the user can be queried over a range of performance levels that should theoretically translate into a range of annoyance/satisfaction levels.

The second approach is user-driven and is implemented with an “*I am annoyed*” button; the user is encouraged to hit a button at any moment when they are unsatisfied with the performance. This mechanism was first proposed with the OneClick framework [4] where it was coupled with a single specific application. Our scenario differs in two ways: (i) OneClick runs at a web server with pre-recorded content and ours runs on the end-host itself; (ii) we do not know a priori which application the user is annoyed with, whereas OneClick runs in the context of a single specific application.

These two approaches have different advantages. ESM will sample more evenly throughout the performance range which is useful because we want to understand the range of user satisfaction across levels of performance. However, we also seek extra samples at moments of performance problems since our end goal is to be able to do diagnosis. We thus need extra questionnaires from these rarer events. The *I’m annoyed* button has the benefit of obtaining more samples at poor performance epochs. We thus elected to incorporate both mechanisms to obtain the benefits of each.

Experience sampling algorithm: The purpose of the experience sampling algorithm is to decide under what conditions and when to pop up the questionnaire to obtain user feedback. One could simply trigger the questionnaire at random epochs. However, as one might guess, and as the end-host data we obtained so far indicates, most of the time a user’s machine is lightly loaded, and thus a purely random scheme results in large numbers of user samples at low load. Although there are many different phenomena that can underlie a performance degradation, we simplify things by using network load as a proxy for performance assuming that higher user annoyance usually co-occurs with a heavily loaded network. It is thus intuitive that a weighted random sampling scheme which weighs more heavily the higher load epochs will yield a more even distribution of samples. (Each sample here refers to a single completed questionnaire.)

The basic form of our sampling algorithm incorporated into HostView is presented in Algorithm 1. The algorithm has six static parameters (α_l , α_m , α_h , γ_l , γ_m , γ_h) and three adaptive parameters (T_l , T_m and *active-time*) that are adjusted per user and updated on a daily basis. The thresholds T_l and T_m denote load ranges in the traffic. The throughput, denoted *throu*, is considered low if $0 \leq \text{throu} \leq T_l$, medium if $T_l < \text{throu} \leq T_m$, and high if $\text{throu} > T_m$. These parameters are computed per user as the 85 and 95 percentile values of the user’s empirical distribution. These parameters are adaptive because they are updated each day. On a given day, the value is taken from the same day of the previous week. If no such measurements exist, we use data from the previous day. The sampling is bootstrapped by setting $T_l = 117\text{Kbps}$ and $T_m = 727\text{Kbps}$; these values were extracted from the data collected in the first pilot study.

The α parameters capture the fraction of time a user

spends in the different load ranges. Since we do not know this a-priori, we make a reasonable guess and set $\alpha_l = 0.85$, $\alpha_m = 0.10$ and $\alpha_h = 0.05$ implying that a user spends 85% (10%, 5%) of the time in a low load state (medium, high, respectively). The γ parameters, γ_l , γ_m , γ_h , are the probabilities of taking a sample when the load is low, medium or high respectively. These are set to 0.02, 0.4 and 0.9. Finally, the parameter N is a polling interval.

Our algorithm works as follows. Every N seconds, the ESM algorithm computes the load and determines whether it is low, medium or high based on the values of T_l and T_m . Depending on the current state, the algorithm computes a random value and compares it to the corresponding γ values. The decision of whether to pop-up the ESM questionnaire is based on the outcome of this comparison and also on whether the user was actually using his machine in the previous minute. Our goal is to obtain a maximum of three completed questionnaires per day; this number was chosen based on the results of our survey [10]. We require that the samples be complete; a deferred sample (users are allowed to postpone the questionnaire), or a timeout (the questionnaire goes away after 6 minutes of the pop up) does not add to this number. Thus the selection of the polling interval N is important.

The choice of N presents a fundamental tradeoff: if N is small and we sample too much, user annoyance increases because either there are too many questionnaires or the questionnaires occur too close in time. On the other hand, if N is too large, we won’t get enough samples to perform diagnostic analysis. We limit the maximum number of questionnaires per day to 4, and design the algorithm to hit an average between 2 and 3 ESM questionnaires per day. There are various heuristic schemes that could be selected to choose N , including static and adaptive approaches. In our pilot study, we elected to use an *Adaptive Polling Interval* scheme that computes $N = (0.102 / 4) * \text{active time}$ where 0.102 is the average probability of taking a sample at any interval, 4 is the maximum number of questionnaires per day, and *active time* is defined as the number of seconds per day the user is actually using the keyboard or the mouse of their computer. Each new day, before the ESM algorithm starts, HostView predicts the active time value for the day based on the same day of last week, or if that is not available, then it uses the active time of the previous day. We set the minimum value of *active time* to be three hours.

When the questionnaire pops up, the user has the choice to answer it immediately, to postpone it, or to ignore it. If it is ignored, the questionnaire will “timeout” and disappear after 6 minutes. If the user answers the questionnaire, then we return to the basic algorithm to select the next time to pop up a the questionnaire. However, if the user postpones or ignores it, then they were be asked again in 30 minutes, on average.

Algorithm 1 Experience sampling algorithm

```
1: Every  $N$  seconds :
2:   if (machine not idle) and (Numsamples <= 3) :
3:     if (throu <  $T_l$ ) and (Rand <  $\gamma_l$ )
4:       take sample
5:     else if ( $T_l$  <= throu <  $T_m$ ) and (Rand <  $\gamma_m$ )
6:       take sample
7:     else if ( $T_m$  <= throu) and (Rand <  $\gamma_h$ )
8:       take sample
```

User	Days	Total pop-ups			Answered surveys			Unanswered surveys		Load in Kbits/s	
		Low	Med	High	Low	Med	High	Deferred	Timed out	85th %	95th %
1	23	21	32	35	9	20	31	2	26	12.57	41.76
2	12	11	11	25	0	2	5	38	2	21.57	286.09
3	18	11	9	23	4	4	9	6	14	22.69	120.34
4	10	5	7	6	2	4	3	2	7	123.73	449.02

Table 2: User sample types

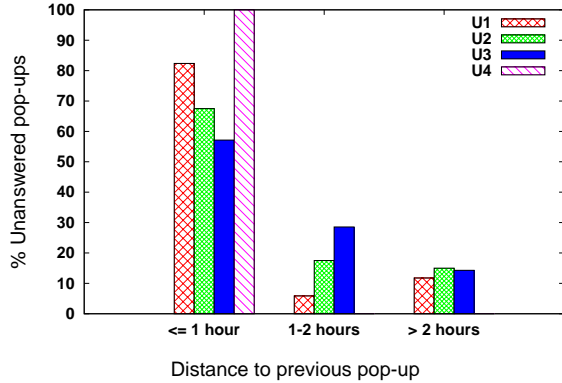


Figure 1: Distribution of unanswered pop-ups.

3.3 Takeaways from pilot study

We conducted a second pilot study of two weeks with four LIP6 students to evaluate the user feedback mechanism. In this experiment, we collect packet headers, application context, system performance logs and user feedback. Table 2 presents the number of ESM samples per user, the number of days each user ran HostView, and the 85th and 95th percentile of the load distribution of each user. A sample is labeled *low*, *medium* or *high* depending on the value of the instantaneous load with respect to T_l and T_m .

We make a few observations from this pilot study. First, we see that a third or more of the samples collected for each user occurred during heavy network load. This reflects our choice of giving a higher weight to sampling heavy traffic episodes than sampling light episodes and confirms correct implementation of the approach. Second, the last two columns of Table 2 justify HostView’s design decision to customize the load thresholds to each user. The distribution of load for User 1 and User 4 are dramatically different; in fact the 85th and 95th percentiles of User 1 are not even in the same order of magnitude as User 4. Third, users behave differently when it comes to dealing with the ESM questionnaire. For instance, User 1 only used the “Ask me later” option to defer the questionnaire twice, but his/her questionnaire timed out 26 times. User 2, however, deferred answering the questionnaire 38 times. This justifies the design decision to include multiple options for answering, postponing, and timing-out the questionnaire because all of these end up being used. Fourth, we observe that the total number of times the questionnaire popped up varies across users. Users 2 and 4 ran HostView for a roughly similar amount of time (12 and 10 days). However, user 2 received 43 pop-ups while user 4 was only queried for feedback 18 times. In this case, the reason for this variation is the behavior of user 2, who only filled in 15% of the pop-ups, which forced the ESM

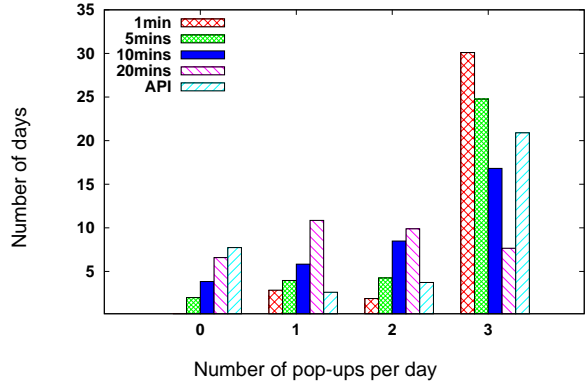


Figure 2: Average number of pop-ups per day (100 simulations with a maximum standard error of 0.15)

algorithm to oversample. In other cases, the variation in the total number of pop-ups is also due to the sampling probabilities (see Alg. 1), the adaptive thresholds and whether the user was actively using her machine one minute before the pop-up.

In our pilot study, some users complained that pop-ups were too close in time. After studying their data, we realized that our predictors for user active time could be quite erroneous and that user active time is not easily predictable from their behavior on the previous day or the same day the previous week. We evaluated the correlation between completed responses and the frequency of pop-ups as an indication of user annoyance. Figure 1 plots the fraction of unanswered pop-ups as a function of the distance to previous pop-up. We observe that when two pop-ups happen within less than one hour, users are less likely to answer them (over 50 % for all users). Therefore, we decided to modify our algorithm to ensure a one hour minimum gap between popping up two consecutive questionnaires.

3.4 Tradeoff: Polling frequency vs. number of daily samples

Because our pilot study showed that it is very difficult to predict a user’s active time, and since we suspect that this complexity may not be necessary in practice, we considered other heuristics for defining the polling interval N . The simplest solution is set it to a fixed time, such as 1, 5, 10 or 20 minutes. Using the data collected in the second pilot, we simulated each of these settings to assess how many questionnaires each of our users would get each day. The simulation for each user utilizes their actual active time, load levels, etc.

Figure 2 plots the daily number of pop-ups for the various fixed- N schemes. We observe that a polling interval of 1

minute yields the highest number of days where we achieve the goal of getting three samples per day. However, a 1 minute interval presents the risk of taking the three samples in a short time duration (a little more than three hours) which means that for people who are very active on their machines, the samples will not be spread throughout the day. We conclude that a polling interval of 5 minutes is a reasonable choice and we incorporate this design decision in the final version of HostView along with the one hour minimum gap between consecutive pop-ups.

4. DISCUSSION

In this paper, we outlined the tradeoffs involved in collecting end-host measurements annotated with user feedback. We spent a considerable amount of time studying and analyzing the overhead of different candidate techniques for collecting end-host data that is complete and sufficient enough for us to study the root causes of performance degradations. The techniques for data collection that we elect to implement are those that have low or moderate overheads and are associated with high utility. We also discussed a candidate experience sampling algorithm and highlighted the difficulties of sampling users because of the unpredictability of their behavior. We described how we used the feedback of our second pilot to modify some of the parameters of our experience sampling algorithm. Our next step is to run HostView for a longer period of time, with a larger set of users, so that we can research characterizations and solutions to end-host diagnostic problems that combine performance and perception.

ACKNOWLEDGEMENTS

The authors would like to thank all participants of the pilot studies. This work was supported by the European Community's Seventh Framework Programme (FP7/2007-2013) no. 223850 and the ANR project C'Mon.

5. REFERENCES

- [1] W. Jiang and H. Schulzrinne, "Modeling of packet loss and delay and their effect on real-time multimedia service quality," in *Proc. NOSSDAV*, 2000.
- [2] K.-T. Chen, P. Huang, and C.-L. Lei, "How sensitive are online gamers to network quality?," *Commun. ACM*, vol. 49, no. 11, pp. 34–38, 2006.
- [3] S. Tao, J. Apostolopoulos, and R. Guérin, "Real-time monitoring of video quality in IP networks," *IEEE/ACM Trans. Netw.*, vol. 16, pp. 1052–1065, October 2008.
- [4] K.-T. Chen, C. C. Tu, and W.-C. Xiao, "Oneclick: A framework for measuring network quality of experience," in *Proceedings of IEEE INFOCOM 2009*, 2009.
- [5] S. Guha, J. Chandrashekar, N. Taft, and D. Papagiannaki, "How Healthy are Today's Enterprise Networks?," in *Proc. of the Internet Measurement Conference*, October 2008.
- [6] E. Cooke, R. Mortier, A. Donnelly, P. Barham, and R. Isaacs, "Reclaiming Network-wide Visibility Using Ubiquitous Endsystem Monitors," in *Usenix Technical Conference*, 2006.
- [7] C. R. Simpson, Jr. and G. F. Riley, "NETI@home: A distributed approach to collecting end-to-end network performance measurements," in *PAM2004*, April 2004.
- [8] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang, "PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-area Services," in *OSDI*, (San Francisco, CA), 2004.
- [9] F. Schneider, J. Wallerich, and A. Feldmann, "Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware.," in *PAM*, 2007.
- [10] D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft, "Perspectives on Tracing End-Hosts: A Survey Summary," in *SIGCOMM CCR*, April 2009.
- [11] "Event Tracing: Improve Debugging and Performance Tuning with ETW." <http://msdn.microsoft.com/en-us/magazine/cc163437.aspx>.
- [12] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Rizzo, and K. Claffy, "Gt: picking up the truth from the ground for internet traffic," in *ACM SIGCOMM CCR*, 2009.
- [13] "Appln. layer packet classifier for linux." <http://17-filter.sourceforge.net/>.
- [14] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet Traffic Classification Demystified: Myths, Caveats, and Best Practices," in *ACM CoNEXT*, 2008.
- [15] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On dominant characteristics of residential broadband internet traffic," in *Proc. of Internet Measurement Conference*, 2009.
- [16] S. Consolvo and M. Walker, "Using the Experience Sampling Method to Evaluate Ubicomp Applications," *IEEE Pervasive Computing Magazine*, vol. 2, no. 2, 2003.
- [17] "Likert scale." http://en.wikipedia.org/wiki/Likert_scale.
- [18] M. Csikszentmihalyi and R. Larson, "Validity and Reliability of the Experience-Sampling Method," *Journal of Nervous and Mental Disease*, no. 175, pp. 526–536, 1987.