Predicting User Dissatisfaction with Application Performance on Home Gateways

Diego Neves da Hora, Maryam Najafabadi, Anna-Kaisa Pietilainen, Renata Teixeira

Predicting User Dissatisfaction with Application Performance on Home Gateways

Diego Da Hora diego.dahora@lip6.fr
Maryam Najafabadi m.mousaarab@gmail.com
Anna-Kaisa Pietilainen anna-kaisa.pietilainen@lip6.fr
Renata Teixeira renata.teixeira@lip6.fr

September 27, 2013

Abstract

Home gateways connect devices in the home to the rest of the Internet. The gateway is ideally placed to adapt or control application performance to better fit the expectations of home users. However, it is not clear if we can infer user (dis)satisfaction with application performance with data available in a home gateway (*i.e.* mostly traffic metrics). Joumablatt et al. used machine learning methods to predict user (dis)satifaction with network application performance at end-hosts. In this paper, we study the feasibility of appying such predictors on a gateway. We show that we can measure the relevant metrics on a home gateway, we improve the proposed non-linear SVM predictor with parameter tuning, and we show that the gateway based predictor can achieve similar performance compared to the end-host predictor.

1 Introduction

Traditional network performance monitoring systems have focused on measuring low level network performance metrics such as delay, jitter, losses and bandwidth and raising alarms when unusally high (or low) values are observed. However, the end user experience of network application performance does not necessarily correlate well with these low level performance metrics. For example, video players typically implement a playout buffer that will mask short transient network problems from the user; or adapt dynamically to the available bandwidth by switching the video stream bit rate. Moreover, mapping raw network metrics to user experience does not only depend on the application: different users may have different expectations for their network performance that may also vary depending on the device or the environment where the application is used.

First steps towards a general mapping of raw network performance metrics to user experience were taken by Joumblatt and al. [6, 5]. The authors implemented first an end-host tool, HostView [6], to collect raw performance metrics, application information and explicit user feedback on network application performance. The tool collected data from 19 users who agreed to run the tool from two weeks up to six months on their personal computers. In [5], the authors apply machine learning methods to build a predictor of user dissatisfaction with

network application performance. The basic idea is to build a predictor that takes as an input low level application performance metrics (including flow level RTT, retransmissions, jitter, throughput; host CPU usage and cross-traffic; and wireless conditions). The predictor is trained with a data set where inputs are explicitely labeled with user feedback (*satisfied* or *dissatisfied*). In the end-host data, a non-linear SVM predictor performed the best in this prediction task.

The end-host point of view is perfect to observe what users are experiencing, but it has some limitations. In particular, the network is a shared resource, and a single host will only have a limited view to the global state of the system. In this work, we will focus on user experience in home networks, and study the feasibility of predicting user experience with data measured on the home gateway. The home gateway is an ideal place to observe the network traffic as it can see all traffic within the home; and measure separately problems occurring inside the home (that may only affect a single device) and on the access link/network (that would impact all hosts). Moreover, the home gateway is well placed to take actions in order to adapt or control application performance to better fit the expectations of home users.

In summary, our work extends the previous work in two aspects: (i) we study the feasibility of a gateway based implementation of the predictor (Section 2), and (ii) we improve the previous predictor by applying a grid-search based parameter tuning for non-linear SVM (Section 3). We find that the majority of the metrics collected by HostView can be collected on a gateway, the only exceptions being strictly end-host based metrics such as CPU or the mapping of flows to application executable names. However, with simple techniques such as well-known port numbers and DNS names, we are able to identify a significant fraction of applications on the gateway. Moreover, on the gateway we can collect more metrics such as home and access link delays; or cross-traffic within the home network. Our predictor tuning confirms the suitability of non-linear SVM to the prediction task, but in contrast to the previous work, we find that the default SVM parameters do not give the best possible performance. In our analysis (Section 4), we emulate a gateway based predictor with the available HostView data and show that the gateway predictor achieves similar performance compared to the end-host predictor.

Despite the improvments in the predictor performance, several questions related to the predictor design remain open. We discuss possible future work directions in Section 5. In summary, there are four main issues to address: (i) we need to collect data from real home gateways to better test and tune a gateway based predictor, (ii) we must better asses the risk of over-fitting with the current choice of parameters, possible solutions include adding more features and users available in the HostView data, (iii) we need a better understanding of the generalization of our predictor, does it apply to any application and any user, or do we need specialized predictors?, and (iv) several challenges remain to be solved in order to use the predictor in a real online user experience monitoring system.

2 Data Collection: End-Host versus Gateway

In this section we first review the data collected by Hostview [6] and the set of application performance metrics we can collect on the end-host. Then, we

	End host	Gateway
Flow metrics	End-to-End RTT End-to-End Jitter TCP Resets TCP Retransmissions	WAN and LAN RTT WAN and LAN Jitter TCP Resets TCP Retransmissions
Application metrics	Application data rate	Heuristic App. data rate
Machine metrics	Host data rate CPU	MAC Address data rate
Wireless metrics	End host SNR	Gateway SNR

Table 1: Metrics available at the end host and at the gateway

discuss which of these metrics we can collect on the home gateway. To train our gateway-based predictor in Section 3, we will need application performance data collected at the home gateway annotated with user feedback, but these datasets are not available. Hence, the last part of this section explains how we can use the HostView dataset to emulate the gateway point of view and what are the limitations.

2.1 End-Host Data Collection

Hostview [6] is a tool to collect network and system performance data annotated with the user feedback on the quality of the connection. The HostView data includes anonymized packet traces, periodic CPU load, periodic wireless signal strength and noise. In addition, Hostview uses the GT toolkit [3] to sample the sockets table every second to assign the application binary to ongoing connections.

The first column in Table 1 summarizes the performance metrics HostView extracts. We can extract the RTT, TCP Resets and TCP retransmissions from packet traces collected by Hostview using tools such as tcptrace [1]. We can calculate Jitter from the timeseries of RTT samples. We use the mapping from application executable names to connections to compute flow metrics and data rate per application. We can extract CPU and SNR metrics from the CPU load and wireless signal logs.

HostView also collects feedback from users with two complementary mechanisms. The Experience Sample Methodology (ESM) samples user perception at most three times a day. The goal of ESM is to sample user experience at a wide range of performance levels. The "I am annoyed" button is always displayed on the screen and users can use anytime to report instances of poor performance. Joumblatt et al. [5] used the user answers to these questionnaires to infer whether users are satisfied with application network performance.

In this paper, we analyze data collected from 19 users who agreed to run HostView from two weeks up to six months. The same dataset used by Joumblatt et al. [5].

2.2 Gateway Data Collection

Our goal is to infer application network performance with control of the home gateway alone. Although the gateway has access to all internet traffic to and

from home devices, we can't compute the exact same metrics as from the end-host. We discuss how gateway metrics differ from end-host metrics for each of the categories in Table 1. The second column of the table summarizes the gateway metrics.

- **Flow metrics.** We can identify all the traffic from the end-host by filtering per mac address. However, the flow performance metrics observed by the end-host are not exactly the same on the gateway. Figure 1 shows the difference between calculating the RTT from the end-host versus the gateway. When measuring the RTT from the end-host, RTT samples from upload traffic measure the RTT from the end-host to the server (LAN RTT + WAN RTT) and RTT samples from download traffic measure the time the end-host takes to process each packet. When measuring the RTT from the gateway, the RTT samples from upload traffic measure the RTT from the gateway to the server (WAN RTT only) and RTT samples from download traffic measure the home network RTT (LAN RTT). The Jitter observed by the gateway also differs from the end host. The end host can only observe end-to-end jitter. At the gateway, we can observe WAN Jitter for upload traffic and LAN Jitter for download traffic. The TCP resets and TCP retransmissions observed by the gateway and the end user are the same.

- **Application metrics.** The gateway doesn't have access to which application executables generate which traffic flows. However, it's still possible to map flows to applications on the gateway using heuristics. The IANA list of well known ports [2] can be used for detecting well-known applications. For example, for all the connections identified by the GT tool as *ssh* in our data, we can correctly identify 99.11% of the bytes using only the IANA list. For *mail*, we can correctly identify 92.48% using the IANA list. However, the IANA list cannot be used to detect applications that don't use standardized port numbers or use shared port numbers, such as Skype and Dropbox.

- **Machine metrics** We do not have access to CPU usage statistics, but we can get host data rates from packet traces, filtering hosts by mac address.

- **Wireless metrics.** If the end user connects to the gateway over wireless, a common setup in home networks, we can have the SNR observed by the gateway. The SNR measured at the end host and the gateway are different, because they are being observed from different points.

2.3 Emulating Gateway Metrics from End-Host Data

We use Hostview data to emulate the traffic observed by the gateway. From the packet traces collected at the end-host we can extract the flow metrics. There's no difference between the TCP Resets and TCP retransmission observed between the gateway and the end host, thus, they can be promptly used to emulate the gateway. Also, we can emulate mac address data rates from the end host packet traces. We can only emulate a topology with the one end host and the gateway, a limitation of this dataset.

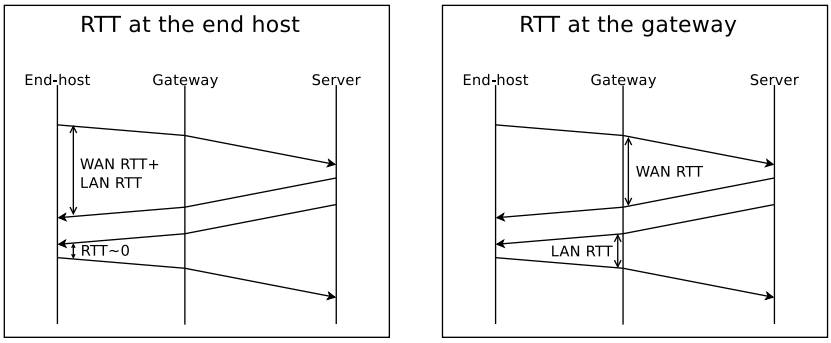


Figure 1: Measuring RTT from different vantage points

We estimate the WAN RTT using the end-to-end RTT and the same follows for Jitter, since it is calculated from RTT samples. Users are affected by end-to-end RTT, thus one might say we are giving the gateway predictor data more related to the user, biasing the results. However, the gateway can estimate end-to-end RTTs very well. We can obtain LAN RTT samples for download traffic, which should be similar across connections. Thus, an estimation of the end-to-end RTT is the average end host LAN RTT plus the WAN RTT. Once we have a gateway deployment, we can compare if it's better to use the estimated end-to-end RTT and Jitter metrics or to use both LAN and WAN RTT and Jitter metrics.

We emulate SNR at the gateway using the end-host SNR. Although SNR measured by the gateway and the client will be different in practice, this difference shouldn't affect our predictors. The predictor uses SNR as computed by a single vantage point and what is meaningful is how SNR varies when measured from this single point. The variation of SNR should be similar when observed from the gateway or the client.

CPU load is not available on the gateway, and should not be used in the gateway classifier. In order to compare our work with previous results [5], we assume that we can detect all connections belonging to the selected set of applications.

3 Building the Predictor

In this section we overview the original non-linear SVM based predictor used to predict user dissatisfaction at end-hosts [5]. We use a similar approach to build our gateway based predictor. In order to be able to compare the performance of end-host versus gateway based predictors, we train the gateway predictor using the same end-host data set excluding metrics that we cannot measure directly on a gateway. We also improve the predictor training by adding a parameter tuning step where we choose the best performing parameter combination using grid-search.

Metric Group	Features	Gateway
RTT	RTT_{mean}, RTT_{median}, RTT_{std}, $\text{RTT}_{95\%ile}$	X
Jitter	Jitter_{mean}, Jitter_{median}, Jitter_{std}, $\text{Jitter}_{95\%ile}$	X
TCP Resets	RST_{total}	X
TCP Retransmissions	RETR_{total}	X
Application data rates	$\text{APP-RATE-UP}_{total}$, $\text{APP-RATE-DOWN}_{total}$	X
Host data rates	$\text{HOST-RATE-UP}_{total}$, $\text{HOST-RATE-DOWN}_{total}$	X
CPU	CPU_{mean}, CPU_{median}, CPU_{std}, $\text{CPU}_{95\%ile}$	-
SNR	SNR_{mean}, SNR_{median}, SNR_{std}, $\text{SNR}_{95\%ile}$	X

Table 2: Feature vectors. Each feature in the table is aggregated over 5-minute and 1-hour bins resulting in 44 features in total. We exclude the CPU based features from the gateway predictor.

3.1 End-host Predictor

Processing raw data for predictor training involves typically three stages: (i) quantization, (ii) feature selection, and (iii) labeling. The end-host predictor designed in [5], quantized data over a short 5-minute interval to capture the state at the time of the problem, and a longer history bin of 1-hour to capture user frustration due to more persistent problems. The features we selected based on prior knowledge of possible problems and their impacts. We summarize the selected features in Table 2. Note that for several metrics, a set of features is created using simple statistics of the distribution of the data over the quantization time bin. Finally, binary labels, *satisfied* or *dissatisfied*, were assigned to each vector.

In order to evaluate the feasibility of a gateway based predictor, we emulate the gateway features using the end-host feature vectors described above with few modifications. As discussed in the previous section, some of the metrics, such as the host CPU, would not be available on a gateway. Hence, we exclude the corresponding features from the predictor. Moreover, some of the available end-host metrics (*e.g.* SNR or RTT) do not necessarily correspond to values an actual gateway would have measured. However, due to lack of data from real gateways, in this work we take the end-host values to be a good enough approximation.

Before training the predictor, all features are scaled to range $[0, 1]$, and to deal with imbalanced data (the HostView data set contains many more vectors labeled *satisfied*), over-sampling of negative vectors is applied. The authors of [5] find that a non-linear SVM classifier with radial basis function (RBF) kernel performs the best in the prediction task. Consequently, we focus on non-linear SVMs in the remainder or this work.

3.2 Predictor Tuning

Considering a dataset with instance-label pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots, l$ where $x_i \in R^n$ and $\boldsymbol{y} \in \{1, -1\}^l$, the SVM classifier solves the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i > 0 \end{aligned} \tag{1}$$

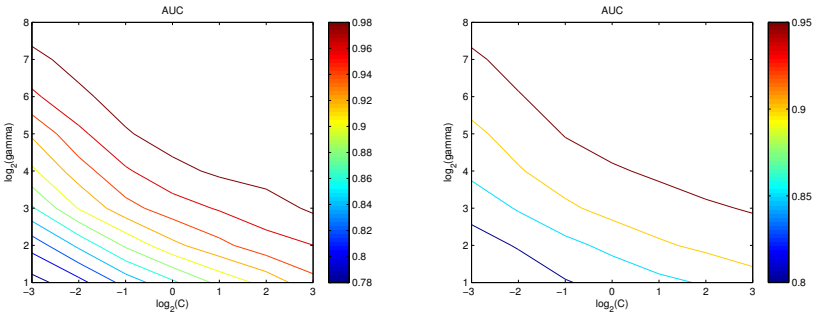
Function ϕ is the function that maps the feature vectors to higher dimensional feature space. SVM finds a linear hyperplane that separates two classes with the maximal margin. $C > 0$ is the penalty parameter for misclassification. To solve this optimization problem we do not need the actual feature vectors in higher dimensional space but their dot products in higher dimensional space is enough for computational purposes. Their dot products in higher dimensional space is given by a kernel function: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$. The RBF kernel function we select is defined as $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp(-\gamma|\boldsymbol{x}_i - \boldsymbol{x}_j|^2), \gamma > 0$.

As can be seen from above, there are two free parameters in an RBF based SVM: C and γ. As we mentioned before, C is the cost parameter for misclassification. For larger values of C, SVM produces a closer fit to the training data. γ controls the flexibility of resulting hyperplane and for small values of γ the decision boundary is nearly linear.

To select the best parameter values for the final predictor, we apply a grid search in a parameter space of C and γ as proposed in [4]. We test various pairs of (C, γ) and evaluate the performance using 10-fold cross-validation.

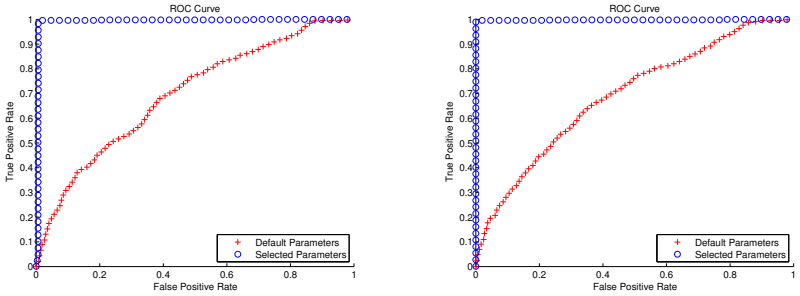
The actual performance metric for selecting the best model depends on the application. In our application the goal is to detect users' dissatisfaction. If we consider the dissatisfied samples as positive samples, it is obvious that we want the true positive rate (TPR) to be high. It means that we want to maximize the number of dissatisfied samples that are correctly detected as dissatisfied. On the other hand, we do not want to detect some satisfied samples as dissatisfied, as in this case the system would raise unnecessary alarms. Hence, we also want the false positive rate (FPR) to be as low as possible. Considering these requirements, we need a performance metric that captures the trade-off between TPR and FPR. For this purpose, we select area-under-curve (AUC). AUC is the area under ROC curve. ROC curve is a graphical plot with TPR on the y-axis versus FPR on the x-axis. The curve is constructed by considering TPR and FPR at different threshold values (SVM does not produce binary labels but some threshold needs to be chosen in order to classify a test vector as positive or negative sample). In general higher AUC values correspond to high TPR and low FPR which we consider good for our application.

In Figure 2 we show the contour plots of the grid search to find the best parameter values for the end-host and gateway predictors (Figures 2a and 2b respectively). The AUC values become higher with higher values values of C and γ. Based on the results we select as the parameter values $(C = 2^3, \gamma = 2^7)$ for both host and gateway predictors.



(a) End-host predictor. (b) Gateway predictor.

Figure 2: AUC for range of values C and γ.



(a) End-host predictor. (b) Gateway predictor.

Figure 3: ROC curves comparing the default ($C = 1, \gamma = \frac{1}{\text{nb. of features}}$) and the selected best parameters ($C = 2^3, \gamma = 2^7$).

The default parameter values, ($C = 1, \gamma = \frac{1}{\text{nb. of features}}$), used in [5] differ considerably from our choice. In Figure 3, we compare the ROC curves for both predictors with the default and the selected parameters. We can see clearly how the parameter tuning improves the performance of the predictor significantly: for most threshold choices we obtain an optimal trade-off between TPR and FPR.

The risk with choosing very high values of C and γ is over-fitting, and hence loosing in generality of the predictor when facing new data. However, in the remainder of this work we consider these parameter values and leave it as future work to test the predictor with new data sets.

4 Comparing End-Host and Gateway based Predictors

In this section we compare the performance of gateway and end-host based predictors to analyse the feasibility of gateway based predictor. In the previous section we saw that the selected parameters result in similarly improved performance for both the end-host and the gateway predictors. We summarize the AUC results in Table 3.

Predictor	AUC (selected parameters)	AUC (default)
End-host	0.9982	0.6938
Gateway	0.9912	0.6909

Table 3: AUC value for predictors based on the selected SVM parameters and the default SVM.

Application	TPR	FPR
Mail	0.91	0.22
SSH	0.90	0.14
Skype	0.70	0.14
Adium	0.68	0.18

Table 4: End-host predictor preformance for specific applications.

We compare also the predictor performance for specific applications. We select four applications that were experimented before in [5] (Mail, SSH, Skype, Adium). For each of these four applications, we divide the related feature vectors in m parts (here we consider $m = 5$). Then iteratively $m - 1$ parts from this specific application are added to samples from all other applications. This new dataset is used to train the predictor and then the predictor is tested for the remaining part. We repeat this for each of the m subsets. Tables 4 and 5 list the true positive and false positive rates for end-host and gateway predictors respectively form this experiment. There are no big differences between the two predictors, hence end-host CPU does not seem to add much information to the predictions. In general, we can see that the TPRs are quite high, and while the FPR are much lower, they remain still at considerably high level. This seems to indicate that a general predictor will need more information about the particular application in order to make correct predictions. We will investigate this further in future work.

5 Discussion

Below we discuss some limitations of this work and potential future work directions. In general, our work suffers from a limited amount of data available, hence, a crucial next step for any further work is to acquire more data both from end-hosts and from home gateways.

Gateway predictor: In this paper we have shown that a gateway based predictor can have a good accuracy, and is in theory feasible in practise. As future work, we plan to collect data from real home gateways to test the gateway predictor with more realistic data. An open problem remains: how to test the predictor performance as the collection of user feedback will be harder with such a system.

Predictor design: Despite the improvements in the predictor performance we achieve with our improved parameter selection, several questions related to the predictor design remain open. Due to the small size of our training and testing data sets, and the small number of negative labels, we cannot rule out the possibility of over-fitting with our choice of the SVM parameters. Hence, we

Application	TPR	FPR
Mail	0.91	0.17
SSH	0.90	0.11
Skype	0.71	0.19
Adium	0.68	0.22

Table 5: Gateway predictor preformance for specific applications.

plan to implement a HostView like tool to collect more data on user experience. In addition, we are currently re-analysing the original data set to include more users and features to the predictor. Some example metrics that seem relevant but were not considered in the original work include more detailed wireless performance metrics such as network utilization, used bit rates and retransmission counts; flow level details such as packet rates, sizes and inter-arrival times; or the environement (at work versus at home).

General predictor: An open design question relates to the generality of the predictor. As our results show, the per application prediction performance is not that good when using a general predictor. Application specific predictors were also found to perform better in [5]. Similarly, we can ask if a general predictor trained with all users' data can efficiently predict the user experience of a previously unknown user? More work remains to be done in order to find a truly general predictor that would work for any application and any user (or to show that no such predictor is possible and that we need more specialized predictors).

Online predictor: The final goal of our work is to build an online monitoring system that can accurately predict user dissatisfaction with networked application performance, and eventually take corrective actions based on these predictions. Apart from building the ideal predictor, many practical challenges remain: how can we collect the feature vectors online either on a gateway or an end-host without incurring too much overhead?, where is the data going to be stored?, where and when should we (re-)train the predictor?, how to make the predictions efficiently in real-time, and how to proceed once a problem is detected?

References

[1] Tcptrace. `http://www.tcptrace.org`.

[2] Iana list of well known ports. `http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`, 2013.

[3] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, et al. Gt: picking up the truth from the ground for internet traffic. *ACM SIGCOMM Computer Communication Review*, 39(5):12–18, 2009.

[4] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. *A Practical Guide to Support Vector Classification*. National Taiwan University, April 2010.

[5] D. Joumblatt, J. Chandrashekar, B. Kveton, N. Taft, and R. Teixeira. Predicting user dissatisfaction with internet application performance at end-hosts. In *INFOCOM, 2013 Proceedings IEEE*, pages 235–239, 2013.

[6] D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft. Hostview: Annotating end-host performance measurements with user feedback. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):43–48, 2011.