NetPrints: Diagnosing Home Network Misconfigurations Using Shared Knowledge

Bhavish Aggarwal
Ranjita Bhagwan
Venkata N. Padmanabhan
Microsoft Research India

Geoffrey M. Voelker
UC San Diego

July 2008

Technical Report
MSR-TR-2008-102

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

NetPrints: Diagnosing Home Network Misconfigurations Using Shared Knowledge

Bhavish Aggarwal[†], Ranjita Bhagwan[†], Venkata N. Padmanabhan[†], and Geoffrey M. Voelker[‡*]
[†]Microsoft Research India [‡]UC San Diego

***Abstract*— Networks and networked applications depend on several pieces of configuration information to operate correctly. Such information resides in routers, firewalls, and end hosts, among other places. Incorrect information, or *misconfiguration*, could interfere with the running of networked applications. This problem is particularly acute in consumer settings such as home networks, where there is a huge diversity of network elements and applications coupled with the absence of network administrators.**

To address this problem, we present *NetPrints*, a system that leverages shared knowledge in a population of users to diagnose and resolve misconfigurations. Basically, if one user has figured out the fix for a problem, we would like this knowledge made available to another user experiencing the same problem. NetPrints records and aggregates configuration information from a large population of clients, annotates it with compact network problem signatures, looks up the appropriate information when a new client experiences a similar problem, and suggests configuration changes to resolve the problem. NetPrints performs all of these steps automatically, with little human involvement. We evaluate NetPrints in the context of several home networking problems actually reported by users, and find that it is effective in sifting through large volumes of shared configuration data to identify the relevant fix.

I. Introduction

A typical network comprises several components, including routers, firewalls, NATs, DHCP, DNS, servers, and clients. Configuration information residing in each component controls its behaviour. For example, a router's configuration tells it who its neighbours are while a firewall's configuration tells it which traffic to block and which to let through. Correctness of the configuration information is thus critical to the proper functioning of the network and of networked applications. *Misconfiguration* can interfere with the running of these applications, leading to user frustration.

This problem is particularly acute in consumer settings such as home networks, where there is a huge diversity of network elements and applications, deployed without the benefit of vetting and standardization that is typical in enterprises, and an absence of network administrators. A home user with a network problem is often left helpless, not knowing which, if any, of a myriad of configuration settings to manipulate.

Nevertheless, it is often the case that a user is not the first one to encounter a problem. Other users may have encountered a similar or the same problem while running the same application with a similar or the same network setup. For example, a particular audio/video chat application may not work with a particular make and model of home router unless the client host is placed on the DMZ. If one user discovers this fix, it can, in principle, be reused by another user faced with the same problem in a similar setting.

Motivated by this observation, we present *NetPrints*, a system that helps users diagnose network misconfigurations by leveraging the knowledge accumulated by a population of users. This approach is akin to how users today scour through online discussion forums looking for a solution to their problem. However, a key distinction is that the accumulation, indexing, and retrieval of shared knowledge in NetPrints happens *automatically*, with little human involvement.

NetPrints comprises client and server components. The client component gathers configuration information from the client host and from network devices such as the home router or NAT box. In addition, it captures a trace of the network traffic associated with an application run and extracts a set of *features* that characterize the corresponding network communication. The client component then uploads its local configuration information along with the network traffic features to the server. In addition, in the case of a failed application run, the user clicks a "help" button to invoke NetPrints diagnostics (this is the only human input needed in NetPrints). This also signals to the server that the configuration information and network traffic features just uploaded correspond to an unsuccessful run of the application. We term the combination of configuration information, network traffic features, and the indication of whether or not an application run was successful an *anecdote*.

The server gathers such anecdotes from clients and constructs a decision tree for every application to represent the *knowledge* of good and bad configurations. It also uses a decision tree algorithm to identify the network traffic features that are important, thereby generating a network *signature* for each application. In addition, the server maintains a *suggestion table* where, indexed by the network signature, it stores a potential set of configuration fixes that other clients have previously reported as their solution to a similar problem.

When the server is presented with a client request triggered by a user invoking "help", it walks down the decision tree that codifies its knowledge and it identifies

[*]The author was a visiting researcher at Microsoft Research India during the course of this work.

configuration changes that might help resolve the problem using a procedure we call *configuration mutation*. If the decision tree traversal does not yield a suitable fix, the server looks up the suggestion table for any isolated configuration changes that might solve the problem. If both the tree traversal and the suggestion table lookup fail in generating a configuration fix, NetPrints infers that the problem is not related to the client's home network configuration.

We have compiled a list of 25 configuration-related home networking problems and their resolutions drawn from online discussion boards, user surveys, and our own experience. Of these, we were able to obtain the necessary hardware and software resources to reproduce 10 problems. All of these problems are amenable to resolution through NetPrints, as we argue later in the paper. However, since we do not have configuration data or network traces from a large population of users, our evaluation focuses on real data gathered for three applications from our testbed, where we artificially varied the network configuration settings to mimic the diversity of settings from an actual population of users. Separately, we evaluate the robustness of NetPrints' decision tree algorithm for different mixes of good and bad anecdotes and when bogus anecdotes are fed to it. Our results demonstrate the effectiveness of NetPrints in diagnosing many different kinds of misconfigurations.

While our focus in this paper is on home network settings, NetPrints could be applied in other settings as well, as discussed in Section V. Also, we focus here on network configuration problems that interfere with specific applications but do not result in full disconnection and in particular does not prevent communication with the NetPrints server. Indeed, such subtle problems tend to be much more challenging to diagnose than full disconnection, which is often the result of a basic problem (e.g., the network cable not being plugged in) or a problem beyond the home network and hence the control of the end user (e.g., a disconnection at the ISP level). Nevertheless, as we discuss in Section V, it would be possible to proactively prefetch and locally cache relevant information from the NetPrints server (e.g., that pertaining to the specific make and model of router in a user's home network), to allow NetPrints to still function in the event of a disconnection from the server.

II. Related Work

Misconfiguration diagnosis has received much attention in recent years. We summarize prior work on problem diagnosis in computer systems and in networks, and discuss how NetPrints relates to it.

A. Peer Comparison-based Diagnosis

There has been prior work on leveraging shared knowledge across end hosts, which provides inspiration for a similar approach in NetPrints. However, the prior work differs from NetPrints in significant ways.

Strider [23] uses a state-based black-box approach for diagnosing registry problems that impact the health of applications on a Windows machine. The basic idea is to trace accesses made to registry keys by an application ("state tracing") and combine this with a comparison of registry settings across machines ("state diffing"), thereby identifying the keys that are the likely culprits. Note that Strider requires a healthy machine to be identified for state diffing. Also, state tracing would be hard to do in the NetPrints context because there are no explicit "accesses" that applications make to network configuration information, whether local or on a separate network device. Rather, the configuration information governs policy (e.g., port-based filtering) that impacts an application's network communication.

PeerPressure [22] extends Strider by eliminating the need to identify a single healthy machine for state diffing. Instead, PeerPressure accumulates registry settings from a large population of machines. Then, under the assumption that the settings on most of the machines is correct, it uses Bayesian estimation to calculate the probability of each registry key setting on the sick machine being the culprit. For each culprit considered in rank order, PeerPressure suggests to the user that the key be set to the most popular value drawn from the population.

The unsupervised learning approach used in PeerPressure has the advantage of not requiring the samples to be labeled, although the system does require the user who is performing the diagnosis to manually identify a sick machine as such. However, this unsupervised approach also means that PeerPressure will necessarily find a culprit. This would be inappropriate in the NetPrints settings, where, for instance, an application failure may be caused by reasons other than local misconfiguration (e.g., disconnection at the ISP). On a separate note, PeerPressure works with discrete valued configuration variables (which makes it hard to deal with ranges, e.g., all settings of MTU less than 1300 may be bad) and moreover assumes that there is only one culprit. The decision tree based classifier we employ in NetPrints avoids these difficulties.

Finally, Autobash [19] helps diagnose and recover from system configuration errors by recording the user actions to fix a problem on one computer and replaying these on another computer that is experiencing the same problem. Autobash uses a set of predicates to generate problem signatures and also to determine whether the speculative application of a solution has in fact fixed the problem. Autobash depends critically on causality tracking with the OS kernel to determine which processes depend on a particular entity (e.g., configuration setting). However, such causality tracking is not feasible in the NetPrints setting since, for instance, the end host does not have visibility into how configuration settings on the home router are being used.

B. Problem Signature Construction

There has been work on developing compact signatures for systems problems that could then be used to look up a database to find previous instances, if any, of the same problem, often having a known solution.

Yuan et al. [26] generate problem signatures by recording system call traces, representing these as n-grams, and then applying support vector machine (SVM) based classification. SVMs allow for very general classification using arbitrary separating hyperplanes. However, it operates on real-valued samples; categorical configuration information (e.g., the wireless security setting, which might be one of WEP, WPA, or none), which NetPrints must accommodate, would first have to be quantified to be amenable to SVM. Instead, NetPrints uses a decision tree based classifier, which can naturally handle discrete as well as continuous configuration parameters, to represent both its knowledge of configurations and application network signatures. More importantly, a decision tree lends itself to easy interpretation, allowing NetPrints to identify of mutations needed to reach a good state from a bad one.

Cohen et al. [6, 7] consider the problem of automated performance diagnosis in server systems. They use Tree-Augmented Bayesian Networks (TANs) to identify combinations of low-level system metrics (e.g., CPU usage) that correlate well with high-level service metrics (e.g., the average response time). A high service response time corresponds to the system being in an unhealthy state. They also present a technique to cluster system states corresponding to similar problems and thereby construct a signature that is then used to look up information (e.g., diagnostic notes) on past instances of the same problem. Their work, unlike NetPrints, focuses solely on quantifiable system metrics. Also, TANs are computationally more efficient that decision trees (which is important when processing large volumes of data in real time), but are not as useful in the NetPrints context since they do not readily identify the mutations needed to reach a good state.

C. Network Problem Diagnosis

There has also been much work on network problem diagnosis. However, there has been a limited amount of work that focuses on misconfigurations in particular.

Active probing has been used to localize faults in wide-area network paths. For example, Tulip [12] probes routers to localize anomalies such as packet reordering and loss on an end-to-end path. Such diagnosis relies on a model of how routers behave. In the context of NetPrints, it may be possible to construct such a model for certain well-understood configuration information (e.g., port-based filters at a firewall), thereby allowing active probing based diagnosis. However, it would be difficult to do so in general because the impact of various configuration settings may not be documented or well-understood. Indeed, our experience has been that the root causes of many home networking problems are firmware quirks or bugs, which would be hard to capture through a model.

In the context of wireless LANs, the broadcast nature of the channel facilitates problem diagnosis based on passive monitoring of network traffic. Examples of such systems include DAIR [3] and Jigsaw [5]. These are again based on model- or rule-based engines, an approach that would be hard to replicate for arbitrary configuration information.

Other systems such as SCORE [10] and Sherlock [2] have modeled, and in some cases automatically discovered, dependencies between higher-layer, observable network events and the underlying network components. These systems then use statistical inference to identify network components that are the likely cause of failures.

WiFiProfiler [4] combines peer cooperation with a rule-based engine to help clients diagnose wireless problems, including several configuration-related ones (e.g., security settings). This system focuses on a setting where the number of peers sharing information is very small (just the nodes within wireless range), hence the dependence on a rule-based rather than a statistical approach.

Finally, there has also been work on using formal methods to check the correctness of network configuration information. For example, the rcc tool [9] includes a constraint verifier that checks for a range of well-understood BGP routing properties.

D. NetPrints Compared to Prior Work

We view NetPrints as being complementary to prior work on network diagnosis in a couple of ways. First, NetPrints focuses on *local configuration* problems at the client end that impact *specific* applications rather than on broad problems that impact the network infrastructure. Second, NetPrints uses a *blackbox* approach appropriate for arbitrary and poorly understood configuration information, avoiding the need for the network behaviour or dependencies to be modeled explicitly. The latter is a key point of difference between NetPrints and the network diagnosis systems targeted at the more organized enterprise networks.

NetPrints draws inspiration from prior work on blackbox techniques to diagnose systems problems and index them with signatures to enable recall. However, NetPrints' goal of identifying how a broken configuration can be mutated to *fix a problem* leads us to use a decision tree based approach, which is distinct from the techniques used in prior work. Furthermore, NetPrints leverages domain-specific knowledge to construct *signatures* of networking problems. So the diagnosis procedure in NetPrints is both state-based and signature-based.

III. NetPrints Design and Implementation

In this section, we describe the design and implementation of the NetPrints system. We begin with an overview of how the NetPrints system operates. We then provide a more detailed description of the design and implementation

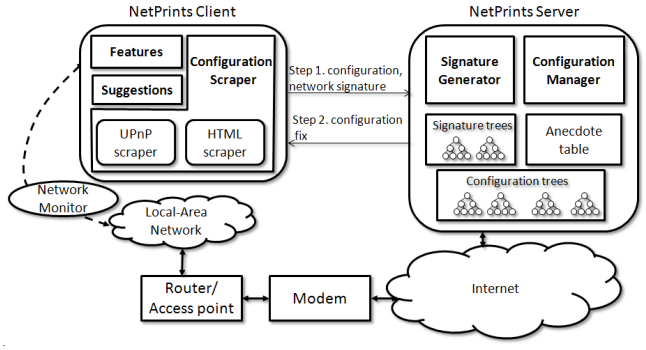


Fig. 1. NetPrints system design

of the various system components at both the client and the server.

To provide context for the challenges of diagnosing home network misconfigurations, we have compiled a list of 25 configuration-related home networking problems and their resolutions drawn from online discussion boards, user surveys, and our own experience. Table I summarizes these problems, their cause, and the configuration changes required to fix them. Glancing through this table shows that many problems affect specific applications, that the causes can be subtle (e.g., no connectivity when STP is not disabled when connected to Comcast networks (#3)), and that the solutions can involve obscure configuration settings, particularly for home users (e.g., the suite of specific settings for the Xbox problem in #22).

A. Operation overview

Figure 1 shows a high-level depiction of the components of NetPrints, both at the client and the server, and how they interact. NetPrints has two modes of operation: the "normal" mode and the "diagnose" mode. In normal mode, the NetPrints client collects information about the normal mode of operation of the user's machine. It periodically determines the set of applications running, a *feature vector* (explained in Section IV-E) that characterizes the network usage of each application, and the home network configuration. It reports a concise representation of this information to the NetPrints server whenever it detects a change since the last time it reported this information to the server. The server, in turn, uses this information to characterize the normal mode of operation for each application reported by the client. We emphasize that NetPrints uses a black-box approach to operating on home network configurations. It does not interpret the semantics of any of the configuration fields and values used to diagnose network problems.

When users experience a problem with a certain application, they click the "help" button on a simple GUI to ask the NetPrints client on their local machine to diagnose the problem. The NetPrints client can identify which application to diagnose automatically (e.g., the application corresponding to the last foreground window before running the NetPrints client) or with the help of the user (e.g., ask the user click on the application window). The client then switches to the diagnose mode, in which it gathers the same information as in the normal mode. However, it labels the information as corresponding to a "bad" state (since there is a problem in running the application) and uploads this information to the server (step 1 in Figure 1). The server compares this "bad" information with the "good" information (represented as a *decision tree*) it has gathered over time from clients that ran this application successfully (i.e., corresponding to the normal mode of those NetPrints clients). The server then reports possible configuration fixes back to the client (step 2 in Figure 1).

In some cases, the server may not be in a position to diagnose the problem based on the configuration information that it has accumulated. This situation could happen because the server has an insufficient volume of samples to be able to distinguish between good and bad configurations (e.g., the application might be new). It could also happen when the problem only impacts a subset of clients, so that the "problematic" configuration actually works well for the majority of clients (e.g., problems #3 and #4 in Table I). In such cases, we rely on *suggestions* to try and resolve the problem. A suggestion is an observation reported by a client that says that a certain configuration change seems to have fixed the problem. The problem is identified using the *network traffic signature* of a problematic run of the application. While not as authoritative as diagnosis based on the decision tree, suggestions can nevertheless be useful in the NetPrints context, just as they are in the human context (e.g., on online discussion boards which discuss problems and potential fixes).

If neither the decision tree nor the suggestion information yields an answer — the configuration reported by a troubled client is not deemed "bad" and there are no suggestions that apply to the application and its network signature — NetPrints will not return any diagnosis or resolution steps. This response is appropriate since it may well be that the problem is unrelated to local network configuration, so any local resolution steps might do more harm than good. This aspect of NetPrints is in contrast to prior systems such as PeerPressure [22], which would necessarily suggest a configuration change even when the cause is not configuration-related.

We will now describe, in more detail, the different components within the NetPrints client and the NetPrints server.

B. The NetPrints Client

The NetPrints client, running as a background process on a computer within the home network, has three principal components: the *configuration scraper*, the per-application *network traffic feature extractor* and the per-application *suggestion generator*.

Configuration scraper

No.	Application	Router	Problem	Cause	Fix
1.	BitTorrent	WRT54GL	Torrents seem to get extremely slow after a while	NAT table filling up too fast	Decrease the NAT table timeout, increase the max. no. of connections in the NAT table
2.	File Sharing	WGT624	Only unidirectional sharing: PC1 is seen on PC2 but not vice versa	Firewall is not properly configured	Allow file sharing through all firewalls
3.	Generic	WRT54GL	Cannot access the network	STP (Spanning Tree Protocol) not supported by Comcast	Disable STP
4.	Generic	Linksys	Cannot access the Internet though the LAN is working	Recent router change, ISP uses MAC authentication, so disallows traffic from the new router	Turn on MAC address cloning
5.	IP Camera	DG834GT	Camera disconnects periodically at midnight, router needs reboot	DHCP problem	Configure static IP on the camera
6.	Online Gaming	WGR614	Disconnected from wireless network immediately or 30 sec into playing	(n/a)	Enable UPnP on router and gaming device
7.	Office Communicator	WRTP54G	Instant Messenger client does not connect from home	DNS requests not getting resolved	Switch off DNS proxy on router
8.	Outlook	WRT54G	Outlook does not connect via VPN to office	Default IP range of router was same as that of the office router	Change the IP range of router
9.	Outlook	WGR614	Router not able to email logs	SMTP server not configured properly	Setup SMTP server details in the router configuration
10.	Outlook	Linksys	Not able to send Outlook messages through Linksys router. Belkin router works fine, Linksys only receives messages	MTU value too high	Reduce MTU to 1458 or 1365
11.	Outlook	WRT54G	Not able to send mail using Outlook	Specific ports not opened on the router	Setup port triggering on router for port 25 and 110 (smtp and pop3 resp)
12.	ROKU	DIR-655	ROKU did not work with mixed b, g and n wireless modes	(n/a)	Change to mixed b and g mode
13.	SSH	WGR614	SSH client times out after 10 minutes	NAT table entry times out	Change router or increase NAT table timeout
14.	SSH server	WRT54G	Not able to setup ssh	Port 22 not forwarded	Forward port 22 to correct IP
15.	STEAM based games	WGR614	Listing game servers causes connection drops	The router misinterprets the sudden influx of data as an attack and drops connection	Upgrade to latest firmware
16.	Streaming Real Media	BEFW11s4	Real streaming kills router	Firmware upgrade caused problems	Downgrade to previous firmware
17.	Streaming media	WGR614	Streaming media is not played	SPI is enabled which drops the connection	Disable SPI in the router configuration
18.	VPN	WGR614	VPN does not work with Cisco VPN Client	Cisco client uses GRE protocol which is not supported with the router	Use a different router
19.	VPN	WRT54G	VPN drops connection after 3 minutes	(n/a)	Set MTU to 1350-1400, uncheck “block anonymous internet request”, “filter multicast boxes” in router configuration
20.	VPN	WRT54G	No VPN connectivity	Old router firmware	Firmware upgrade
21.	VPN server	WRT54G	PPTP server behind NAT does not work despite port forwarding enabled on required ports and PPTP passthrough allowed	IP of server is 192.168.1.109, which is inside default DHCP range of router. Router sometimes is not able to port forward to these IPs inside default range of router	Use static IP outside DHCP range for server
22.	Xbox	WRT54G	Xbox does not connect and all games do not run	Some ports are blocked and NAT traversal is restricted	Set static IP address on Xbox and configure it as DMZ, enable port forwarding on UDP 88,TCP 3074 and UDP 3074, disable UPnP to open NAT
23.	Xbox	WRT54G	Xbox works with wired network but not with wireless	WPA2 security is not supported	Change wireless security feature from WPA2 to WPA personal security
24.	Xbox	WGR614	Not able to host Halo3 games	NAT settings too strict	Set Xbox as DMZ
25.	Xbox	WRT54G	2 Xboxes behind same NAT don’t run simultaneously	Router can’t forward traffic on one port to two different Xboxes	DMZ one Xbox and port forward the other for ports 88 UDP and 3074 TCP and UDP

TABLE I. RECENT CONFIGURATION-RELATED PROBLEMS IN HOME NETWORKS.

The configuration scraper collects three kinds of information:
a) *Network identification information* from the local host running the client: specifically, whether it is using the wireless interface, the wired interface, or both, and whether it is using a static IP address.
b) *Internet Gateway Device identification information*, namely the make, model and firmware version of the device, which in most cases is a home router although in some cases it could be a DSL or cable modem[1]. The scraper obtains this information using the UPnP interface that is supported by many Internet Gateway Devices [20]. It also uses the UPnP interface to obtain the URL for the web interface to the device.
c) *Network-specific configuration information from the device*. The scraper uses both the UPnP interface and the web interface that most routers and modems provide to glean configuration information such as port forwarding and triggering tables, MTU value, VPN passthrough parameters, DMZ settings and wireless security settings. On the routers we tested, the port tables from the web page and the port tables from the UPnP interface were not kept consistent with each other. Consequently, we had to scrape and combine the tables from both interfaces. Some router firmware versions also allow us to scrape the maximum NAT table size and the per-connection timeout for each table entry. These fields can be particularly useful in diagnosing problems such as #1 and #13 in Table I.

While the UPnP interface gives us access to only device-identifying parameters and the UPnP port forwarding and port triggering tables,[2] the web interface is richer but not standardized across routers. In particular, there is no standardized way for parsing the HTML to extract the (key,value) pairs defining the configuration. Consequently, the configuration scraper uses several heuristics to extract configuration information from the router web pages. While these heuristics work across a set of Linksys, Netgear, and D-Link routers in our testbed, it is difficult to know how well these will extend to other routers.

An alternative to parsing the HTML is to leverage the observation that each configuration web page of the device is typically an HTML form that includes a "submit" button. We can invoke this button programmatically on each configuration web page (for example, using the WebBrowser .NET control on Windows). Doing so causes the submission of an HTTP POST request containing all of the (key,value) pairs in an easy-to-parse form. For example, the body of the POST request might contain: `submit_button=index&change_action=&submit_type=&action=Apply&dhcp_start=100&dhcp_num=50&dhcp_lease=1440`. It is straightforward to extract the various DHCP-related configuration settings from this string.

[1]Several ISPs supply home users with modems that include routing, NAT and DHCP functionality.

[2]The UPnP specification for Internet Gateway Devices has many functions defined as optional. In our experience, these functions are usually not supported by typical home router firmware.

Network traffic feature extractor

The network traffic feature extractor characterizes the network usage of each application running on the client machine. In our current implementation, it uses the winpcap library and the IPHelpers API on Windows to tie all observed network traffic to the individual processes, and hence applications, running on the client machine. For each running application, it extracts a set of features by examining its network activity. These features form the *feature vector* for the application. Table II lists the set of features we extract in the form of rules. For every application, the feature extractor creates a seven-bit feature vector. If *at least* one connection of an application satisfies any of these rules, the corresponding bit in the feature vector is set. Note that while all of the features we consider at present are binary, the feature set could be expanded to include non-binary features.

No.	Feature description	Evaluation Type
1.	TCP:Three SYN no response	per-connection
2.	TCP:RST after SYN	per-connection
3.	TCP:RST after no activity for 2 mins	per-connection
4.	TCP:RST after some data exchanged	per-connection
5.	UDP:Data sent but not received	per-four-tuple
6.	Other: Data sent but not received	per-IP address pair
7	All: No data sent or received	overall

TABLE II. NETWORK TRAFFIC FEATURES USED IN NETPRINTS.

We identified the set of features in Table II based on empirical observations of the ways in which an application's network communication may typically fail. The first four features in the table capture various kinds of TCP-level issues that we commonly see in malfunctioning applications. Several applications and services such as multimedia streaming, DNS and VPN clients use transport protocols other than TCP. For all of these, the lack of connectivity in one direction often indicates a networking problem. Consequently, we have included features #5 and #6 to capture the behavior of such applications. Feature #7 characterizes a total loss of connectivity for an application using any transport protocol; problems #12 and #23 in Table I, for instance, are scenarios in which our system would use this feature.

The NetPrints client associates the feature vector extracted from network traffic with the corresponding application generating the traffic. We use a hash of the application binary image as a unique application identifier. The gathered network configuration and the per-application network features form the basic unit of information, called the *anecdote*, that the NetPrints server uses to perform automatic diagnosis of misconfigurations, as we discuss next.

Note that the client generates the network traffic feature vector during the execution lifetime of an application. This

situation would be problematic for applications that are long running; for example, a web browser could remain open on a client machine and be used for days or weeks. One way of constructing the feature vector would be to consider the network activity of the application just within relatively short windows of time. We defer further consideration of this issue to future work.

Suggestion generator

NetPrints uses the suggestion generator to help recommend configuration fixes for new problems that new applications or routers may pose, or to solve obscure configuration problems such as #3 in Table I. When the user reports a problem that NetPrints cannot immediately solve, the suggestion generator on the NetPrints client starts tracking the configuration and the affected application using the configuration scraper and the network traffic feature extractor.

If it perceives a change in configuration (likely entered manually) and, within a pre-defined time window, the application's networking problem disappears, NetPrints infers that the configuration changes fixed the application's problem. It then creates a *suggestion* containing the application binary hash, the network traffic feature vector, and the configuration fix, and uploads this suggestion to the NetPrints server. The server uses these suggestions to generate configuration fixes that may not yet be captured in the decision tree of configurations. We describe this aspect of diagnosing later in Section III-C.

Client issues

Extracting the network traffic feature vectors for an application requires capturing its traffic. One possibility is to do this continuously. This approach has the advantage that both successful and unsuccessful runs of an application would be captured automatically. In our implementation, we split the network signature generator into two parts: a lightweight, continuously running component to capture selected packet headers and connection-to-process bindings, and a relatively more CPU-intensive component that creates the feature vector from the trace. This approach leads to low overhead. We tested our implementation of the continuously-running NetPrints client on a 1.8 GHz laptop PC running Windows Vista Enterprise while streaming video over the Internet and simultaneously synchronizing email folders with the server. We found that the NetPrints client only had a 0.8% CPU overhead under such a scenario.

An alternative would be to monitor applications when they start and to capture traffic only when the application is one for which we have not already extracted the feature vectors since the last change in network configuration. While this alternative would further reduce the overhead, it would mean that when there is a failure of an application for which the feature vector has already been recorded (from a previous successful run) and users click "help", they would have to run the application again for NetPrints to capture traffic from the failed run.

C. The NetPrints Server

As shown in Figure 1, the NetPrints server has two major components: the *configuration manager* and the *network signature generator*, each of which operates on a per-application basis. The configuration manager tracks configuration information from successful and unsuccessful runs of an application. When presented with a misconfiguration, it suggests changes to be made to the (bad) configuration to move it to a good state. We call this step *configuration mutation*. The network signature generator prunes the network signatures uploaded by clients to the minimum set of features needed to characterize and differentiate between the different ways in which an application may fail.

Decision trees

NetPrints uses decision trees as a basis for performing configuration mutation. A decision tree is a predictive model that maps observations (e.g., a client's network configuration) to their target values or *labels* (e.g., "good" or "bad"). Each non-leaf node in the decision tree corresponds to an attribute of the observation and the edges out of the node indicate values that this attribute can take. Thus, each leaf node corresponds to an entire observation and carries a label. Given a new observation, we start at the root of the decision tree, walk down the tree, taking branches corresponding to the individual attributes of the observation, until we reach a leaf node. The label on the leaf node identifies configurations as good or bad.

There are several algorithms for decision tree learning, i.e., for inducing a decision tree from labeled training data. We chose a widely-used algorithm, C4.5 [16], which builds trees using the concept of information gain. The idea is to start with the root, and at each level of the tree choose that attribute to split the data which reduces the entropy by the maximum amount. The result is that the branch points (i.e., non-leaf nodes with multiple children) at the higher levels of the tree correspond to attributes with greater predictive power, i.e., those with distinct values or value ranges corresponding to distinct labels.

When the training data is noisy (e.g., it contains mislabeled samples) or there are too few samples, there is the danger that the above algorithm will over-fit the training data. To address this concern, decision tree algorithms like C4.5 also include a pruning step, wherein some branches in the tree are discarded so long as this does not result in a significant error with respect to the training data (a process called generalization). C4.5 uses a confidence threshold to determine when to stop pruning. In our implementation, we use the default threshold. A consequence of pruning is that if the number of samples is insufficient, these will not be reflected in the decision tree. We evaluate this issue in the context of NetPrints in Section IV-E.

A decision tree has two key properties. First, it enables classification of observations that include both quantitative

and categorical attributes. For example, the decision tree in Figure 6 includes quantitative attributes such as the WAN MTU and categorical attributes such as the security mode. Second, a decision tree is amenable to easy interpretation. It not only enables classification of observations, it also helps identify in what minimal way an observation could be *mutated* so as it change its label (e.g., from "bad" to "good"). With a decision tree, NetPrints can walk up the tree until it hits a branch point that includes a leaf with the desired label as a descendant, and then walk down the tree to that leaf node. For example, in Figure 4, the mutation needed for a WGR614 router to move to a "good" state, with stateful packet inspection (SPI) disabled, would be to enable SPI.

The above properties make decision trees attractive in the context of NetPrints compared to alternatives such as SVMs or Bayesian classification. Both configuration management and network signature generation require the ability to work with quantitative as well as categorical attributes. Furthermore, configuration management can benefit from the mutation recipe that decision trees provide.

Configuration manager

The configuration manager uses the configuration information submitted by clients to learn and construct *per-application configuration trees* using C4.5. Note that configuration information submitted when the user clicks the "help" button is labeled as "bad"; otherwise, it is labeled as "good".

Figure 4 shows an example of such a configuration tree that we generated for the Microsoft Connection Manager VPN application [13] using configuration information from clients using two different models of routers: the Linksys WRT54G and the Netgear WGR614v5. We note that the `pptp_passthrough` attribute (corresponding to whether PPTP pass-through is enabled) is the clearest, even if not a perfect, indicator of whether a configuration is good or bad. So it is at the root of the decision tree.

Algorithm 1 shows the algorithm that the configuration manager uses to suggest suitable configuration changes to the client. With this algorithm, NetPrints uses the subtree from a branch of the nearest parent node for searching for a path that ends in a good configuration. The configuration fields along this path are the candidates for moving the configuration from a bad state to a good state. We chose this approach because it results in minimal configuration changes for solving the problem (the alternative path shares a long common prefix with the original configuration). Alternative algorithms exist, such as traversing to the root of the tree and searching in the subtree of another branch of the root. One issue with that approach is that it might recommend more drastic changes first (e.g., change the firmware or use a different router) instead of simpler configuration changes. Our experience has been that this algorithm works well with the decision trees we have experimented with. Further experience with NetPrints will suggest whether the algorithm needs revising.

Algorithm 1 Configuration mutation algorithm to move from a bad state to a good state.

```
1: sub find_good_conf(bad_leaf)
2: parent_node = parent(bad_leaf_node)
3: good_leaf_node = find_good_leaf(parent_node)
4: conf_fix = traverse_tree(parent_node, good_leaf_node)
5: return
6: end sub
7: sub find_good_leaf(node)
8: if is_good_leaf(node) then
9:    return node
10: else
11:    for all  child_node of node do
12:       good_leaf_node = find_good_leaf(child_node)
13:       good_leaf_node
14:    end for
15: end if
16: end sub
```

Network signature generator

The server also constructs *per-application signature trees* to reduce the network traffic feature vectors submitted by clients down to the most significant features. The signature generator again uses the C4.5 algorithm for this purpose. Figure 5 shows the signature tree generated for the Microsoft Connection Manager VPN application. Of all of the network features classified as good or bad, the signature tree structure shows that only two important features are sufficient to capture all the networking problems that the Connection Manager application sees in our experiments.

Diagnosis procedure

We now describe the server operations in normal mode (corresponding to successful application runs) and in diagnosis mode (corresponding to application failure leading to the user invoking "help") in more detail. In Sections IV-C and IV-D, we illustrate these server operations through examples.

Normal mode: When a client uploads a good configuration and the corresponding application network traffic feature vector, the configuration manager and the signature generator use this to train the configuration tree and the signature tree, respectively. Currently, the decision tree algorithm we use does not allow for incremental training of the trees, hence we use a cache of configurations to perform the training at each step. However, incremental update based algorithms exist [21] and we plan to evaluate these in future work.

Diagnosis mode: When a client uploads a presumed-to-be-bad configuration along with a malfunctioning application's network feature vector, the configuration manager and the signature generator at the server again use this information

to train their respective trees. In addition, the server proceeds to diagnose the problem.

As the first step towards diagnosis, the server traverses the tree top-down, using the presumed-to-be-bad configuration. If this traversal ends at a leaf node that is labeled as "bad", the configuration manager uses Algorithm 1 to find an alternate path from the root to a leaf node that is labeled "good". It uses this alternate path to generate a set of configuration changes that it then conveys back to the client for presentation to the user.

However, as noted in Section III-A, it is possible that the top-down traversal of the tree in fact ends at a "good" leaf node. Such a case can arise if (a) the problem that the client has encountered is relatively new (e.g., because it involves a new application) and so has had an insufficient volume of training samples reported for it to have been incorporated in the configuration tree, (b) the problem only impacts a small subset of clients (e.g., problem #3 with STP and Comcast in Table I) so that the same configuration is reported as good by the majority of clients, [3] (c) the configuration information being reported by the clients is not rich enough, or (d) the failure is not due to local misconfiguration.

The NetPrints server constructs and maintains a *suggestion table* to address cases (a) and (b). The suggestion table is populated with the *suggestions* contributed by clients (Section III-B) and indexed by the network signature of the application *before* the suggested fix was applied. Since the accumulated volume of suggestions would keep growing, the server only remembers a small, fixed number of the most recent distinct suggestions for any given application network signature.

In the event that the configuration information submitted by a complaining client is found to be "good", the server uses the network signature submitted to look up the suggestion table. If one or more suggestions is found, it returns these to the client.

If the suggestion table also does not return an answer, NetPrints declares that it is unable to diagnose the problem. As noted in Section III-A, this result would be appropriate in some cases since the problem may not be related to local configuration at all (case (d) above). However, if in fact the problem is that some critical configuration information is not being captured by NetPrints (case (c)), this would require the client-side scraper to be augmented to extract this additional information.

IV. Experiments

In this section, we describe the experiments we performed to evaluate how effective the NetPrints system is in automatically diagnosing misconfigurations. Our experiments recreate problem scenarios for a set of applications and then verify that NetPrints automatically catches all of the misconfigurations that cause these problems. We first outline our experimental methodology, and then we demonstrate how NetPrints builds suitable configuration and signature trees and identifies configuration mutations, using anecdotes generated for three example applications. Finally, we show that the decision trees that NetPrints produces are robust to changes in the accuracy, skew, size, and mix of the input anecdote set.

[3]One could address this issue by including the ISP name as part of the configuration information, but we avoid doing that since this is non-local information that may not always be discoverable by the client host in an automated manner.

A. Methodology

Using three routers, the Linksys WRT54G, Linksys WRTP54G and Netgear WGR614, we recreated some of the problems specified in Table I. We outline these recreated problems in Table III. However, for the experiments reported here, we focus on three applications — the IIS FTP server, the Microsoft Connection Manager VPN client, and the Xbox 360 gaming console — and two routers — Linksys WRT54G and Netgear WGR614 — to study NetPrints' ability to detect and correct misconfigurations. We specifically pick these example applications because problems related to services running behind NATs, VPN clients, and gaming systems are reported as significant pain-points on the forums we have tracked, as reflected in Table I.

We ran each application in different home-networking environments created by varying the home router (Linksys WRT54G or Netgear WGR614), the type of medium used (wired or wireless), and the configuration settings on the router. From various home router forums such as the Netgear [15] and Linksys [11] help forums, Microsoft support web pages [14], and third-party firmware forums [8], we created a list of typical configuration parameters that users modify on their routers. We determined the exact parameter names and automated the process of varying their settings, using the HTTP POST mechanism explained in section III. We list the details of these configuration parameters below, with the values that each parameter was set to shown in parentheses.

1) MTU size (1100, 1200, 1300, 1400, 1500). Both the Linksys and the Netgear routers used the variable `wan_mtu` to specify this. We limited changing MTU size to these specific round numbers because, in most cases, users do not set the MTU to an arbitrary number under 1500.
2) VPN-specific passthrough fields (on or off). These parameters were available only on the Linksys router. It used three binary variables for VPN-based filtering: `ipsec_passthrough`, `pptp_passthrough`, and `l2tp_passthrough`.
3) Stateful Packet Inspection (SPI) firewall (on or off). This parameter was available only on the Netgear router through the `disable_spi_firewall` binary variable.
4) Wireless security parameters (disabled, WEP, WPA or WPA2). The Netgear router used the `security_type` variable to specify the type

No.	Problem	Router	Bad configuration
1.	Connection Manager fails to connect	Netgear WGR614	SPI firewall disabled
2.	Connection Manager fails to connect	Linksys WRT54G	pptp_passthrough disabled
3.	FTP connections fail to an FTP server behind a NAT	Linksys WRT54G, Netgear WRT54G	No DMZ set
4.	Office Communicator (IM) did not connect	Linksys WRTP54G(VoIP)	DNS proxy enabled
5.	Remote Desktop Connection fails	Linksys WRT54G	No port forwarding enabled
6.	SSH connection times out after 10 minutes	Netgear WGR614	NAT table timeout too short (10 minutes)
7.	SSH connection times out after 30 minutes	Linksys WRT54G	NAT table timeout too short (30 minutes)
8.	Xbox 360 does not connect to Xbox Live	Linksys WRT54G, Netgear WGR614	MTU $<$ 1365
9.	Xbox 360 does not connect to the wireless network	Linksys WRT54G	WPA2 turned on
10.	Xbox 360 does not detect an open NAT	Linksys WRT54G, Netgear WGR614	UPnP turned off

TABLE III. PROBLEMS WE RECREATED

of wireless security and the Linksys router used `SecurityMode`. Also, the Netgear router did not support WPA2.

5) DMZ (on or off). Both routers used the variable `dmz_enable`.
6) UPnP (on or off). Both routers used the variable `upnp_enable`.

In our experiments, we varied the these configuration settings on the router. For each setting, we ran the application and used the NetPrints client's network feature extractor to create the feature vector for the application. We combined this with the configuration information to create an anecdote for the run. When the application worked as expected, we labeled the anecdote as "good". For the runs in which the application encountered networking problems, we labeled the anecdote as "bad". We used this collection of anecdotes to generate the results presented here.

For the evaluation in Sections IV-B, IV-C and IV-D, we input all of these anecdotes to the NetPrints server's configuration manager and network signature generator. These then generate configuration trees and signature trees that, as we show, capture all of the problems that we saw with these specific applications. Our experiments in Section IV-E use the same collection of anecdotes to vary the proportion of good and bad configurations, the diversity in the configuration information, and data set size to show NetPrints' robustness in generating the correct configuration and signature trees and in identifying suitable configuration mutations to fix problems.

B. IIS FTP Server

Usually, people set up FTP servers behind their NATs so that they and people they know can have easy access to data on their local computers from a remote location. The forums showed that a number of people complain about their service not running as expected behind a NAT. To emulate this situation, our first evaluation of NetPrints was on the IIS FTP server [1] running behind a NAT. While varying the configuration, we used a remote FTP client to connect to this server. When we saw the connection fail, we labeled the anecdote as bad. All other anecdotes were labeled good. In this experiment, we used 128 distinct anecdotes, with 64 each labeled as good and bad.

Figure 2 shows the configuration tree that NetPrints generates using these anecdotes. The configuration tree clearly

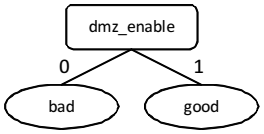


Fig. 2. The configuration tree generated by the configuration manager for the IIS FTP server.

captures the fact that when the variable `dmz_enable` was set, the FTP server worked. Therefore, for any new anecdote for this FTP server that is labeled as bad, the configuration mutation would involve changing the `dmz_enable` field from 0 to 1.

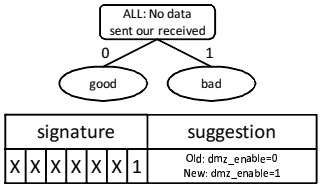


Fig. 3. The signature tree and the suggestion table entries generated by the signature generator and the configuration manager for the IIS FTP Server.

Figure 3 shows the signature tree that the signature generator built using the anecdotes. The differentiating feature in this case is feature no. 7 in Table II. The server determines the problem signatures to enter into the per-application suggestion table by traversing the tree from root to every bad leaf. It sets the value of all the features that it does not see during the traversal as a don't care value, "X". In this experiment, since the FTP server ran into only one kind of problem, the server generated only one signature, with all values except the 7th value set to X. If and when a client reports that a particular configuration change fixed a problem (i.e., it makes a suggestion), with a feature vector matching this signature, the server makes an entry in the suggestion table indexed by the signature.

C. Microsoft Connection Manager

The Microsoft Connection Manager [13] is a PPTP-based VPN client. To collect anecdotes with the Connection Manager, we varied the router configuration and, for each configuration, we tried to connect our client to a VPN server. If the VPN connection was successful, we labeled

the anecdote as good, and if the connection did not go through, we labeled the anecdote as bad. We collected 360 anecdotes using the Linksys router, of which in 120 cases the VPN client connected successfully to the server, and 120 runs using the Netgear router, of which the VPN client connected successfully to the server 60 times.

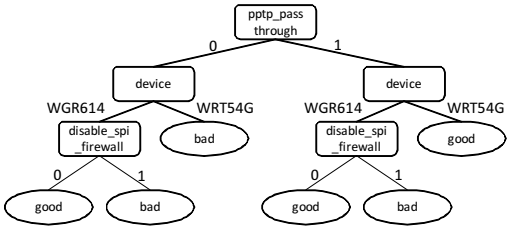
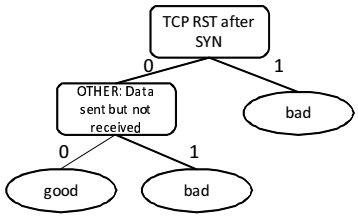


Fig. 4. The configuration tree generated by the configuration manager for the Connection Manager VPN client.

Figure 4 shows the configuration tree for the Connection Manager application that the NetPrints server generated. Of all the configuration parameters, the configuration manager picked `pptp_passthrough`, `device`, and `disable_spi_firewall` as the discerning configuration parameters. Therefore, from the anecdotes, NetPrints *automatically* identifies that the connection manager fails to connect through the Netgear WGR614 router if `disable_spi_firewall` is turned on, and it fails to connect through the Linksys WRT54G router when `pptp_passthrough` is disabled. The configuration mutation step will therefore suggest changing bad configurations by changing `pptp_passthrough` from 0 to 1 or changing `disable_spi_firewall` from 1 to 0, depending on the router used. This matches with the three problem scenarios that we manually reproduced for the VPN client, shown in Table III.



signature							suggestion
X	1	X	X	X	X	X	Old:disable_spi_firewall=1 New:disable_spi_firewall=0
X	0	X	X	1	X	X	Old:pptp_pass=0 New:pptp_pass=1

Fig. 5. The signature tree and the suggestion table entries generated by the signature pruner and the configuration manager for the Connection Manager VPN client.

Figure 5 shows the signature tree that the NetPrints signature generator creates for the Connection Manager. Of the seven features shown in Table II, only two features appeared to be most discerning — "TCP:RST after SYN" and "OTHER: Data sent but not received". It turns out that the feature vectors in the bad anecdotes that the Netgear WGR614 router generates almost always have the former feature set, while a large percentage of the bad anecdotes that the Linksys router generates have the latter feature set. The C4.5 algorithm, therefore, automatically extracts these in the signature tree. The signature generator uses the signature tree to generate signatures for every problem. these signatures are then used as indices into the suggestion table. Figure 5 shows the two signatures that are created and used to index the suggestion table.

D. Xbox 360

Xbox Live [25] is a service that allows Xbox users to play single-player and multi-player games, chat, and interact over the network. When the game "Halo 3" released, we found a large amount of activity on the different forums discussing home networking issues that hindered online multi-player gaming with Xbox 360 and Xbox Live. Consequently, we decided to evaluate how the Xbox 360 interacted with the Xbox Live service under different routers and router configuration settings.

One problem that arose during this experiment was that we could not run the NetPrints client's feature extractor directly on the Xbox since it is not user-programmable. Xbox Development kits are available at a much higher price than consumer Xboxes, so the NetPrints client could, in principle, be ported to the Xbox. However, for the sake of these experiments, we emulated a NetPrints client on the Xbox by instead running the client on a PC that is able to monitor all of the Xbox's network communication. In the wired network scenario, we set the PC to be in bridge mode, placed in between the Xbox and the home router. For the wireless case, we used a PC, with a wireless interface set in monitor mode, to sniff all packets to and from the Xbox.

We collected 147 anecdotes with the Linksys WRT54G and 100 anecdotes using the Netgear WGR614 router while varying the configurations. 50 of the former and 50 of the latter were good anecdotes. Our methodology to determine whether the Xbox was suitably connected to Xbox Live was to run the "Test Live Connection" tool from the Xbox Dashboard. This tool checks that the Xbox 360 is connected to the network, either via a wired or a wireless connection, and that it has a valid IP address and a DNS server setting. It uses a specific test server to check whether the home router handles ICMP messages as expected, and to check the MTU value of the router. If any of these tests fail, the tool reports an error. The test also classifies the NAT on the router as one of "open", "moderate" or "strict", depending on the port assignment policy and the port filtering policy of the NAT [24]. Xbox Live users prefer to have an open NAT because this gives them the maximum functionality and highest performance while playing online games. We therefore label any anecdote for which the tests fail, or for which the NAT is labeled as "moderate" or "strict", as bad.

Figure 6 shows the NetPrints server's configuration tree generated using the Xbox's anecdotes. There were three

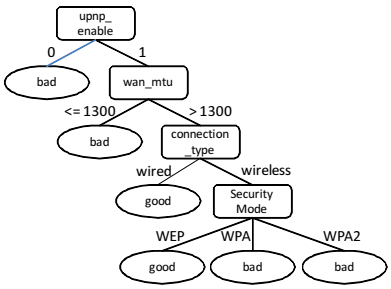
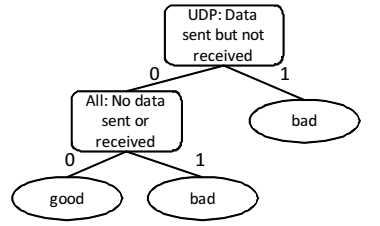


Fig. 6. The configuration tree for the Xbox 360 configuration data.

misconfigurations that the configuration manager learned from the anecdotes. First, to make the NAT open,the router needs to enable UPnP. Second, the Xbox requires the MTU value set to be set to greater than 1300 for it to able to connect to Xbox Live. Third, the Xbox wireless adapter could not connect to a wireless network if the security mode used was WPA2.

NetPrints' findings are the same as the configuration fixes we manually generated and show in Table III, except for the MTU fix. We found information both on the Xbox dashboard tool and on the support pages that the Xbox Live service requires the MTU to be set to 1365 or higher for a connection to succeed. However, given that in our experiments we set the MTU to one of five round numbers, we could not make a more informed choice than setting the MTU size to larger than 1300.

This experiment also shows that the NetPrints server can capture and solve a mix of different kinds of configuration issues: a general error (upnp_enable needs to be on), a service requirement (wan_mtu needs to be set higher than 1365), and an unsupported feature (WPA2 not supported).



signature							suggestion
X	X	X	X	1	X	X	Old:MTU=1200 New:MTU=1500
X	X	X	X	0	X	1	Old:SecurityMode=WPA2 New:SecurityMode=disabled

Fig. 7. The signature tree and the suggestion table entries generated for the Xbox 360 network signatures.

Figure 7 shows the signature tree that NetPrints generates for the Xbox. Although NetPrints detected three problem configuration values in this experiment, the signature tree appears to capture only two features as problematic: ALL:no data sent or received, and UDP:Data sent but not received. It turns out that our feature vector did not capture the difference between having the NAT in moderate or strict mode and having it in open mode. (Indeed, this

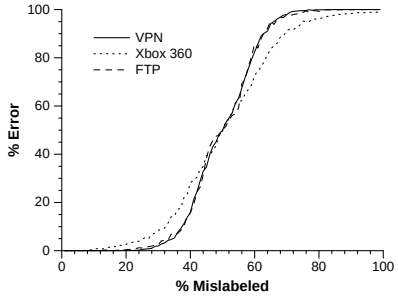


Fig. 8. Sensitivity of the decision trees to mislabeled configuration data.

configuration setting only has a bearing on functionality such as hosting games.) While this would impact our ability to use the network signature to index the corresponding suggestions, the configuration tree (which in any case is NetPrints' first line of defence) nevertheless captures the relevant configuration information (`upnp_enable`).

E. Robustness tests

Finally, we perform a series of experiments to evaluate the robustness of the configuration decision trees to various conditions not found in the experimental data sets we generated. These conditions include mislabeled configurations submitted to the server, skewed distributions of configurations, the responsiveness of the decision trees to adapt to configurations for new problems, and the sensitivity of the decision trees to the balance of good and bad configurations at the server.

1) Mislabeled configurations: The configuration snapshots in the data sets presented in Section IV-A and used in our experiments thus far are all correctly labeled as to whether they represent good or bad configurations for a particular application. In a deployed system, however, configurations added to the shared repository on the server will not always be labeled correctly. Mislabeling could happen for various reasons, including application failure due to reasons other than local misconfiguration, a user not choosing to invoke NetPrints' "help" when an application fails, [4] or the user deliberately invoking "help" even when the application works fine.

Mislabeled configurations could potentially lead to troubleshooting a problem incorrectly, such as identifying a bad configuration as a good one. Even with mislabeled configurations, though, we would still like the decision trees to tolerate the mislabeling and troubleshoot user configurations correctly.

To evaluate the sensitivity of our configuration decision trees to mislabeling, we performed the following experiment. For each application in our workload, we started with the known, correct set of labeled configurations and their associated decision trees. We then chose a random

[4]Note that this would normally have no impact since no anecdote would be uploaded either. However, if it so happens that the client has not already reported an anecdote for the application in question since the most recent configuration change (and hence will do so now) *and* the application fails *and* the user fails to invoke "help", only then will a mislabeling happen.

percentage $_p$ of those configurations and mislabeled them, flipping their labels from good to bad and vice versa. From this set containing mislabeled configurations, we again generated decision trees and compared them with the original trees generated using correct labels. To account for variability in the random choice, for each percentage $_p$ we performed multiple trials of the experiment each with different random mislabelings.

If the two trees are equivalent across all trials, we consider the configuration set to be completely tolerant to $_p$% mislabeling. Basically, in this case, despite the mislabeled data, the C4.5 algorithm produces a pruned decision tree that makes the same decisions as the original tree. As a result, the configuration mutation procedure will remain as accurate as with correctly labeled data — using this tree will still correctly identify all configuration fields that could change a bad configuration to a good one, even though the tree was generated with some mislabeled data.

If the two trees diverge, the decision trees begin to overfit the data as a result of the mislabeling. In this case, the decision trees incorporate features of the mislabeled configurations and the configuration mutation step might report incorrect fields back to the user. Using the tree formed with the mislabeled set of configurations, we can count the number of original configurations incorrectly classified and use this as an error metric.

Figure 8 shows the results of this experiment on the configurations for the VPN and Xbox 360 applications. The x-axis shows the percentage of mislabeling of configurations, and the y-axis shows the fraction of configurations correctly labeled in the decision tree based upon the mislabeled configurations. For each data point we performed 100 trials and present the average across the trials. The results indicate that the applications are fairly resilient to mislabeling. The decision trees for the VPN, Xbox, and FTP applications completely tolerate mislabeling (0% error) when 15%, 2%, and 2% of the configurations are mislabeled, respectively. With 1% error, in which configuration mutation can report an incorrect configuration fix for 1 out of 100 diagnoses, the applications are very tolerant to mislabeling: in this case, the VPN, Xbox, and FTP applications tolerate 28%, 13%, and 23% mislabeling, respectively. When more than 30% of configurations are mislabeled, though, the resulting decision trees begin to overfit substantially.

Those familiar with evaluation of learning techniques will immediately recognize that our methodology is not performing cross-validation on the data with training and testing sets. The reason is that we are not using the decision trees as classifiers. In other words, NetPrints does not use decision trees to classify or predict whether a configuration is good or bad — all configurations from the client already have labels (“good” or “bad”) associated with them. The mislabeling experiment performs an extrinsic evaluation of the problem in terms of the utility of identifying an appropriate configuration mutation for a diagnosis in the face of incorrect labels.

2) Skewed configuration distributions: The configurations in our raw data sets are roughly uniform in distribution in terms of the settings of the various parameters. In practice, however, some configurations are likely to be much more prevalent — have higher skew — than others. (For example, UPnP might be disabled on 90% of the routers.) In particular, one bad configuration is likely to have high skew for a given problem: a default configuration for a device, with an incorrect setting for a parameter, preventing an application from working. Similarly, the resulting working configuration, with the parameter setting corrected, is likely to have high skew as well.

Does configuration skew further change the sensitivity of the decision trees to mislabeling? For each application in our workload, we chose two configurations representing a default bad configuration and a default good configuration. We then introduced duplicates of those defaults to create skew. For an experiment with $_q$% skew we created duplicate default configurations such that the sum of default good and bad configurations comprised $_q$% of all configurations. In a set of 1000 configurations with a skew of 20%, for example, 100 will be the same good configuration and 100 will be the same bad configuration (200 total). We then performed the mislabeling experiment above, varying skew from 0–95% with 100 random trials per skew value.

We found that skew does not affect the sensitivity of the decision trees to mislabeling. For all of the applications and skew values, the effect of mislabeling was the same as with the original distribution of configurations.

3) Responsiveness: Over time the configuration decision trees need to be responsive to new problems experienced by users. With each new application, NetPrints will generate a separate new decision tree. For existing applications, though, users will encounter new problems with existing devices, and new devices will arrive on the market with their own set of unique problems. Ideally, the decision trees will be highly responsive to these new problems and adapt quickly upon encountering configurations that correspond to them. Until the decision trees adapt, however, NetPrints has to rely upon items in the suggestion table for resolving problems, as noted in Section III-A.

We evaluated the responsiveness of the decision trees by simulating a scenario where a new device arrives on the market and users experience new problems with the device. We simulated this scenario by creating an initial decision tree based only upon the Netgear WGR614 router. We then incrementally added configuration reports for the Linksys WRT54G router until the decision tree adapted to represent configuration problems on the new router. We randomly chose configurations to add from the set of Linksys reports, and we repeated the experiment 50 times for each application. We simulated the situation where NetPrints receives equal numbers of bad and good reports (the problem gets fixed as well).

Table IV shows the results of these experiments. For each

Application	# Configurations
VPN	4
Xbox 360	17
FTP	0

TABLE IV. RESPONSIVENESS TO NEW PROBLEMS.

application, it shows the number of configurations added until the decision trees adapted in all random trials of the simulation. The responsiveness of the decision trees varies across applications because the different applications start with different initial trees. For the VPN application, the decision tree adapts very quickly after seeing a handful of configuration reports for the problem in the new router. The Xbox 360 takes longer to adapt, and the FTP result is expected since both routers experience the same problem: the initial tree already captures it and no adaption is necessary.

Finally, there is a tension between being responsive to new problems — extending the trees — and generalizing in the face of mislabeled data — simplifying the trees. The confidence threshold of the pruning algorithm (Section III-C) could be tuned to alter the balance between the two.

4) Good versus bad: Without experience of a live deployment, we can only speculate on the balance of good and bad configuration reports at the server. It is useful to know how sensitive the decision trees are to the ratio of good and bad configurations to determine whether NetPrints should react to unusual situations.

To explore this sensitivity, for a given application we randomly selected from its set of configurations and varied the ratio of good to bad selections. We also varied the total number of configurations selected from 200–20,000. For each set of parameters we again made 100 random trials. Only in the extreme cases (almost all were good or bad) were the decision trees sensitive to the ratio. As a result, we conclude that the ratio of good to bad configurations will not impact the quality of the decisions trees in practice.

V. DISCUSSION

In this section, we outline several issues that a large-scale deployment of NetPrints could potentially face and the ways in which NetPrints could be extended to solve a larger set of configuration-related problems in diverse environments.

Will people use a service like NetPrints? While we cannot measure the frustration that users feel while trying to manually troubleshoot a home networking issue, we believe that a service like NetPrints has tremendous potential of being useful to an ever-growing population of home-networking users. Moreover, NetPrints imposes very low overhead (Section III-B) and is not intrusive on users. Furthermore, in-home networking using media-based devices [17, 18] are making the home network even more complex and difficult to troubleshoot. Such heterogeneity will only increase the need for a system like NetPrints.

Privacy. As with any system that collects information from users, privacy is important. We ensure the privacy of an individual NetPrints user by ensuring that we do not collect any identifying information, such as the PPPoE login and password, the ISP name, WAN IP address, the DNS server addresses, or the wireless SSID. None of the configuration fixes that the NetPrints server proposes do not involve any of these sensitive fields. Furthermore, the network signatures derived from tracing the network capture only very high-level information (e.g., three TCP SYNs without a response) and no raw packet data.

Proactive and reactive operations. Currently, NetPrints works reactively, i.e., it only solves a problem if a client reports it. An alternative approach is to be more proactive in providing configuration changes to the client. Given the knowledge of the set of applications running on the NetPrints client, the NetPrints server could anticipate potential configuration-related problems and suggest preventive configuration changes.

NetPrints in other environments. While our system targets the home network, the NetPrints design methodology is equally applicable to other settings such as large enterprise networks that have different kinds of networking devices, configurations and network topologies. We therefore plan to test the efficacy of the NetPrints approach in the enterprise network setting.

VI. CONCLUSION

We have described the design and implementation of NetPrints, a system to automatically troubleshoot home networking problems caused by router misconfigurations. We use a decision tree-based learning algorithm to aggregate anecdotes from multiple clients and create a concise representation of good and bad configurations. For every reported problem, NetPrints uses the configuration tree or suggestion table to generate suitable configuration fixes. Using three example applications we demonstrate that NetPrints is able to diagnose a variety of misconfigurations and that it is robust to changes in the size, mix and skew of the configuration information.

REFERENCES

[1] Microsoft Internet Information Services (IIS). http://www.microsoft.com/windowsserver2003/iis/default.mspx.

[2] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards Highly Reliable Enterprise Network Services Via Inference of Multi-level Dependencies. In *SIGCOMM*, 2007.

[3] P. Bahl, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill. DAIR: A Framework for Managing Enterprise Wireless Networks Using Desktop Infrastructure. In *HotNets*, 2005.

[4] R. Chandra, V. N. Padmanabhan, and M. Zhang. WiFiProfiler: Cooperative Diagnosis in Wireless LANs. In *MobiSys*, 2006.

[5] Y. Cheng, P. B. John Bellardo, A. C. Snoeren, G. M. Voelker, and S. Savage. Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis. In *SIGCOMM*, 2006.

[6] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *OSDI*, 2004.

[7] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, Indexing, Clustering, and Retrieving System History. In *SOSP*, 2005.

[8] DD-WRT Forums. http://www.dd-wrt.com/phpBB2.

[9] N. Feamster and H. Balakrishnan. Detecting BGP configuration faults with static analysis. In *NSDI*, 2005.

[10] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. IP Fault Localization via Risk Modeling. In *NSDI*, 2005.
[11] Linksys forums. http://forums.linksys.com.
[12] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet Path Diagnosis. In *SOSP*, 2003.
[13] Microsoft Connection Manager. http://support.microsoft.com/kb/221119.
[14] Microsoft Support Website. http://support.microsoft.com.
[15] Netgear forums. http://forum1.netgear.com.
[16] J. R. Quinlan. *"C4.5: Programs for Machine Learning*. Morgan Kauffman, 1993.
[17] The ROKU SoundBridge Network Music Player. http://www.rokulabs.com.
[18] The Slingbox: Watch TV in Another Room or Another Hemisphere. http://www.slingmedia.com/go/slingbox.
[19] Y.-Y. Su, M. Attariyan, and J. Flinn. Autobash: Improving configuration management with operating system causality analysis. In *SOSP*, 2007.
[20] Universal Plug and Play Internet Gateway Device Specification. http://www.upnp.org/standardizeddcps/igd.asp.
[21] P. E. Utgoff. Incremental Induction of Decision Trees. *Machine Learning*, 4:161–186, 1989.
[22] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic Misconfiguration Troubleshooting with PeerPressure. In *OSDI*, 2004.
[23] Y.-M. Wang, C. Verbowski, J. Dunagan, Y. Chen, H. J. Wang, C. Yuan, and Z. Zhang. STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support. In *LISA*, 2003.
[24] Xbox NAT Type Detection. http://www.xbox.com/en-US/support/connecttolive/xbox360/connectionmethods/troubleshootliveconnection-testnat.htm.
[25] The Xbox Live Service. http://www.xbox.com/en-us/live/.
[26] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma. Automated Known Problem Diagnosis with Event Traces. In *EuroSys*, 2006.