DYSWIS: Crowdsourcing a Home Network Diagnosis

Kyung-Hwa Kim*, Hyunwoo Nam*, Vishal Singh†, Daniel Song‡, and Henning Schulzrinne*
*Columbia University, New York, New York
†Inncretech, Princeton, New Jersey, ‡Rice University, Houston, Texas

***Abstract*—Existing failure diagnostic techniques for end users are insufficient to pinpoint the root causes of network failures due to their limited capabilities to probe other network elements. We present DYSWIS, an automatic network fault detection and diagnosis system for end-users. DYSWIS leverages user collaboration to distinguish important network faults from false positive indications, and diagnoses the root cause of the fault using diagnostic rules that consider diverse information from multiple nodes. Our rule system is specially designed to support crowdsourcing and distributed probes. We have implemented DYSWIS and compared its performance with other tools to prove that several network failures which are difficult to be diagnosed by the single-user probe can be detected and diagnosed successfully with our approach.**

***Keywords*-Network troubleshooting, Crowdsourcing, Network failure, Distributed system**

I. Introduction

While operating systems and computers have generally become more user-friendly and reliable, Internet usage can still be frustrating – applications fail silently, things that worked yesterday do not work today, and failures are often transient.

Compared to a few years ago, consumer Internet usage has changed in at least four aspects: Users now expect to connect to a wide variety of networks, from home and office networks to Wi-Fi hotspots and cellular networks. Applications have become more demanding, as almost every application, from online calendars to games, relies on Internet connectivity and a number of applications, such as Voice over Internet Protocol (VoIP) and video on demand (VoD), require consistent performance. Finally, such applications often rely on the proper functioning of up to half a dozen parties, from the local wireless network to DNS servers, content delivery networks (CDNs), and various middleboxes. For all of these components, professional assistance is either unavailable or expensive, with the result that most users need to become unwilling network administrators (or rely on their technically-savvy children or friends for assistance). Thus, users have no good way to know whom to call or what to try when things go wrong.

Most applications provide, at best, minimal support to help pinpoint potential sources of trouble. For example, if Web access is slow, the cause could be high packet loss on the local wireless network due to interference, an overloaded residential Internet connection, wide-area network problems, a misconfiguration in the Network Address Translation (NAT) box, or a remote server problem. The appropriate action differs in each case, ranging from using a third-party DNS server to simply waiting and hoping that the server recovers.

Motivated by these real-world problems, we have developed DYSWIS ("Do You See What I See"), a system for end users and enterprises to diagnose a range of network-related problems. DYSWIS differs from other approaches in relying on the assistance of other network users, modeling the common pattern where one person asks somebody close by, "Hey, is your Internet working?" Reflecting the proliferation of services, both standardized and proprietary, DYSWIS is designed to be extensible by users and third parties, such as vendors of applications. New probes and rule sets can be added to the running system.

Thus, DYSWIS is the first complete system that automatically diagnoses common network problems for end users, using peer assistance and an extensible probing and a rule framework.

The main contributions of the DYSWIS architecture are:

- **Automatic detection of significant failures** We propose a mechanism that leverages end-user collaboration to distinguish meaningful failures from false positive faults among a number of detected faults in the end-user environment.
- **Optimized design for crowdsourced rules** To support crowdsourcing of network experts effectively, we designed a rule system that is composed of small independent rules. Also, DYSWIS provides a simple application interface that enables multiple groups of developers, network administrators, and application vendors to participate in writing new probe modules and diagnostic rules.
- **Practical design for distributed probes** A node categorizes other nodes by their properties. Then, a node compares probing results from different networks and reasonably infer the status of the network infrastructure, which is normally invisible to end-users, without any help from network core devices.
- **Design for distributed network** DYSWIS is specially designed to support decentralized networks such as distributed hash table (DHT), which enables nodes to collaborate without an infrastructure and achieve Internet scale. As each node's information and fault statistics can be published as key-value pairs to a DHT, other nodes can discover

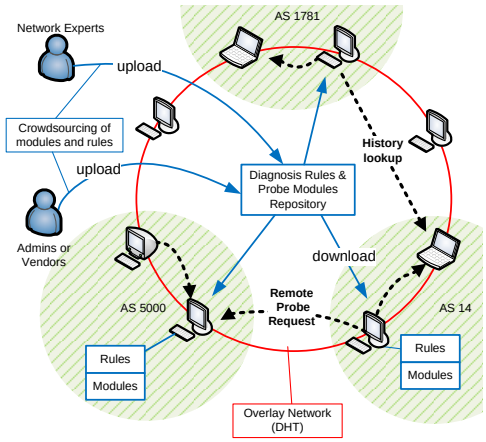


Figure 1: DYSWIS Architecture

appropriate nodes effectively.

From Section II to Section IV, we present the architecture and mechanisms of DYSWIS. In Section V, we evaluate our approach, and in Section VI, we discuss several security issues.

II. Overview

DYSWIS is designed to be a framework that supports diagnostic software running on end users' machines, such as desktops and laptop computers. We present how DYSWIS supports end users and how its crowdsourcing approach enables end users, developers, and network administrators to contribute new rules and diagnostic modules to expand DYSWIS functionality.

A. Peer network

The key feature of DYSWIS is the collaboration of end users. Therefore, a node first needs to discover other users who are willing to assist in the problem diagnosis. A centralized server can be used to maintain a list of available peers, and an alternative method is to build a distributed overlay network (e.g., DHT) composed of end users, which is more scalable and tolerant to the single-point-of-failure problem [1]. Our current implementation adopts the latter approach.

Since we are focusing more on partial network faults, we assume that a node can connect to the DYSWIS network (a central server or other DHT nodes), or can at least have a list of other nodes cached from when the network was available. Service discovery technologies such as Bonjour [2] can also be used when a DYSWIS node cannot join the DYSWIS network while the local area network is available.

1) Peer classification and discovery: Since DHT systems only support exact-match lookups, a DYSWIS node publishes node information with multiple key-value pairs. For example, a single node can be represented by multiple keys such as an Autonomous System Number (ASN), a subnet, an IP address, or whether it uses a NAT. The value corresponding to the key contains the node's IP address, port number, and properties such as type of operating system (OS), network connection (e.g., Ethernet or wireless), or whether it uses a firewall. Table I describes several examples of the keys. These key-value pairs are registered to the DYSWIS network and enable other users to discover appropriate nodes easily.

Table I: Node types and examples of DHT keys

Node	Format of keys	Example of keys
Sister	NAT@[IP address]	"NAT@128.59.21.16"
Near	public@[subnet]	"public@128.59.16.0/24"
Internet	public	"public"
Far	public@[subnet]	"public@AS22"

We categorize peers into five groups according to where they are located. We define following node types:

- ***Local node***: A node currently diagnosing faults.
- ***Sister node***: A node behind the same NAT device.
- ***Near node***: A node within the same subnet.
- ***Internet node***: A node located in any other subnets.
- ***Far node***: A node located in the service provider network of a remote server (e.g., a web server).

To discover a specific type of collaborating nodes, a node queries the DYSWIS network with a corresponding key. For example, to obtain a *near node*, the key must include the subnet information or address of the first-hop router. If the local node is behind a NAT, we often need to discover a *sister node* to obtain the view from the same environment. In this case, the key includes the public IP address of the NAT device. To seek an *Internet node*, we simply query with a key, "public", that returns a list of random nodes from other networks. We can filter out *near nodes* from the list to obtain only *Internet nodes*. In addition, we can discover a *far node* located at a specific subnet. This *far node* is useful when we need to obtain the information from the subnet at which a remote server is located. Some examples of keys are listed in Table I.

III. Detecting Faults

We implement two methods for detecting a network problem automatically: packet monitoring and application plugins.

A. Monitoring packets

DYSWIS monitors raw network packets and checks various failure conditions. First, we check whether the response packets contain error indicators such as "name not found" in DNS, "404 not found" in HTTP, or an RST flag in TCP packets. Then, we monitor the no-response situations. For example, we check if there are responses to outgoing requests such as TCP SYN packets, DNS queries, or HTTP GET messages. If there is no response to these packets, DYSWIS reports it as a problem. Finally, we track the number of TCP retransmissions and duplicated ACKs to examine the status of the current network performance.

Table II: Examples of false positive failures and applications

False positive failures	Applications
mDNS packets	Bonjour
HTTP long polling	Facebook, Dropbox, Gtalk
TCP Retransmission	Video streaming, file download
TCP RST packets	Video streaming (YouTube)

This monitoring approach enables us to detect a number of hidden failure symptoms without the assistance of other applications. However, we discovered that many of these failure indicators occur as part of normal application-specific mechanisms. In this paper, we define a *false positive failure* as a problem indicator detected by the packet-level monitoring but not an actual failure when application-specific behaviors are considered. We describe several examples and present an automatic filtering mechanism in the following paragraph.

*1) **Filtering false positive failures**:* Monitoring packets on real end-user machines, we periodically observe a number of multicast DNS (mDNS) packets that contain a "no such name" error. Although this is a failure message of a DNS query, it is expected if the OS uses the DNS Service Discovery (DNS-SD) protocol to discover services. This happens when a machine sends a service query message, but the service does not exist in the network. In this case, there is no point in reporting these errors to end-users. Another example is HTTP long polling [3]. Long polling is one of the push technologies, which is used by many applications and web sites to communicate interactively with clients without disconnecting a TCP connection. With long polling, a web server does not respond immediately after receiving an HTTP GET request but rather responds after a period of time (e.g., one minute) to maintain the connection. Although this delay is intended, its pattern is identical to the case of a slow response due to poor network performance or a problematic web server. Therefore, even though it is not an actual failure, long polling will be considered as a failure (i.e., high latency) in the packet monitoring system. Another example is TCP retransmission. Although the occurrence of a large number of TCP retransmissions indicates significant performance degradation, it is normal to have a small number of retransmissions caused by temporary network congestion. Thus, a fault detection system needs to set off an alarm only when an unusual number of TCP retransmissions have been detected. In addition, some TCP RST packets can also be misidentified as failures. Although TCP RST packets usually imply a session is unexpectedly terminated, they are intended in some applications. For example, it is known that YouTube may cause a number of TCP RST packets when a client changes video resolution while watching a video [4]. Table II summarizes the examples of false positive failures and applications that cause them.

However, it is impractical to configure every false positive failure scenario beforehand because not all of them are known. For example, it is impossible to know a list of web sites that are using long polling. In addition, it is difficult to set up a threshold of TCP retransmission or TCP RST counts because this depends on applications, protocols, and web sites. It is also possible that other scenarios that we are not aware exist.

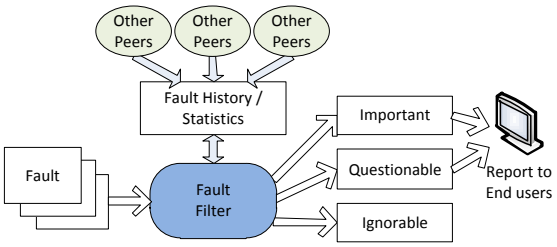


Figure 2: Fault filtering mechanism

In DYSWIS, instead of configuring all the exceptional cases and threshold parameters manually, we filter out the false positive and less important failures using an automatic judging system that uses other peers' failure ratios. We define the failure ratio as the number of failures per packet in a session. For example, if one retransmission has occurred within five TCP packets, then the failure ratio of TCP retransmission is 20%.

DYSWIS periodically publishes these values to the DHT and other peers use them to estimate the significance level of their own failures. In other words, the peers in the DYSWIS network collect failure ratio samples from multiple peers and compare them with their local failure ratio to determine whether the failures are actual problems or not. The collected ratios from other nodes are called the global failure ratio.

Figure 2 illustrates the filtering architecture and Figure 3 describes the detailed flow that estimates the significance level using the global failure ratio. We group the failures into three levels: *important*, *questionable*, and *ignorable*. When a failure is detected, DYSWIS first queries the local cache of failure history. If an insufficient number of samples are retrieved or the samples are stale, it will query the DHT and update the local cache. After that, we calculate the average and standard deviation of the samples. If the local ratio is higher than the average of the global samples, then we consider this failure as important or questionable, depending on the degree. The notable situation is when the failure ratio is similar to or less than the average of the global samples. This means that many other peers have observed the same types of failures as frequently as the local machine has. In this case, it is possible that either the failure is not significant or that other peers have been suffering from the same problem. We can distinguish these situations by observing the timestamps of the failures reported by other nodes. If they have been observed constantly over time, then we consider the problem as ignorable (e.g., every node constantly observes many failures with applications that use the long polling technique). On the contrary, if the failures

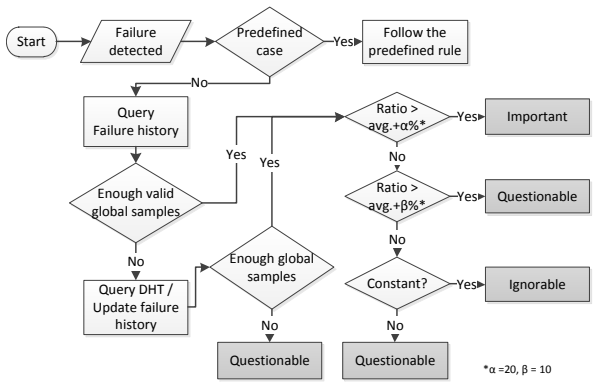


Figure 3: Fault filtering flow

were all observed very recently, then we consider this failure as questionable because it is reasonable to infer that the problem has actually occurred and it can be significant.

If no global failure ratio of a particular failure exists, it implies that nobody has reported this failure before. It is possible that the application (or website) that caused the failure is not popular enough for anyone to have used it. We do not have sufficient clues to judge this case; therefore, we mark this failure as important and report it to the user. This case shows that a lack of event history is one of the limitations of passive probing approaches. This is why our diagnostic process uses active probing that works dynamically, as described in Section IV.

B. Application plugins

The second method to detect network failures uses failure reports from applications. Since applications can observe their failures directly, if they report descriptions of failures to DYSWIS, we do not need to parse captured packets to obtain problem information. For example, if a web browser fails to connect to a DNS server or a particular web server, it reports the problematic server address and failure symptoms to DYSWIS and request a problem diagnosis. Then, the browser can receive back the diagnostic result from DYSWIS and show it to the user. If an application supports plugin development, this approach can be implemented as plugins without modifying the application itself. To prove the feasibility of this concept, we implemented[1] a Google Chrome application that interacts with DYSWIS to report network failures and a Chrome extension that detects performance degradation of YouTube video streaming.

IV. Diagnostics

The second phase of DYSWIS diagnoses the filtered network faults. In this phase, we actively diagnose faults in real time to avoid relying on stale information. The history of faults obtained from other nodes is helpful for filtering the faults, but it is often useless in identifying the root cause. For example, a result of probing that was performed an hour ago has no significance if a failure occurred five minutes ago. In addition, if nobody has tried to connect to the target server during that period, it is difficult to collect proper data to diagnose the fault. In the following sections, we elaborate the probing process and introduce our crowdsourcing-based parallel rule-matching system.

A. Probing modules

Probing modules are small programs that investigate various networking conditions. Each DYSWIS node has a set of probing modules, which can be updated via the module repository.

A probing request contains a name of a probing module to be invoked and fault information to be used as parameters. The response can be either a return value generated by the probing module or 'no response,' which means that the node does not answer. Sometimes, 'no response' provides an important clue for diagnosing the fault. For example, if a node can contact some *near nodes* while failing to contact every *far node*, we can guess there is a network connection issue from the local subnet to the outside network.

B. Crowdsourced rules

The diagnostic rules specify which probing modules need to be invoked in which order and where (local or remote). Their roles also include analyzing the feedback from other nodes and providing final diagnostic results to users. A decision tree is a straightforward way to formulate diagnostic rules. Such a tree indicates which probing module should be invoked, and its result decides the next step of probing. This is repeated until a leaf of a tree is reached, which is either a conclusion or the execution of another rule. We use our own Python-like syntax to represent the decision trees.

However, although decision trees show the diagnostic flow clearly, they do not fit a crowdsourcing approach. For example, our prior work [5] diagnoses VoIP failures using decision-tree-based rules that are designed carefully by VoIP experts. Ironically, however, these complete and large-size rules are not easily upgraded or expanded by other experts because the rules are too intricate to be completely understood. It is very common that the entire decision tree does not work as originally intended when a single part of the rules is modified. Furthermore, more importantly, a decision tree-based-rule can mislead if one of the probes in the middle of the decision tree returns incorrect information. In addition, it often takes a long time to complete the entire decision process since a next step is chosen only after the current probe is completed and the result is returned. If a collaborative node does not answer quickly due to the probing process itself or to network latency, the steps are entirely suspended. For these reasons, we suggest a rule

[1]Chrome app: http://dyswis.cs.columbia.edu, Chrome extension: http://dyswis.cs.columbia.edu/youslow

Table III: Possible causes of connectivity errors and diagnostic rules

Problem ID	Description
C1	Misconfiguration on the user's computer
C2	A problem on the link to a router
C3	Misbehavior of the local router
C4	ISP outage
C5	Link between the ISP and the Internet
C6	Remote service provider network outage
C7	Remote server down
C8	The service provider blocks your ISP
C9	The server blocks your ISP
C10	The service provider blocks your IP address
C11	The server blocks your IP address

(a) Possible causes

Rule ID	Requesting probing to:	Probing module	If response is:	Likely cause	Unlikely cause
R1.1	Sister node	TCP connection	Yes	C1	C2–C11
R1.2	Sister node	TCP connection	No	C3–C11	C1, C2
R1.3	Sister node	TCP connection	No response	C2	-
R1.4	Near node	TCP connection	Yes	C10, C11	C1–C9
R1.5	Near node	TCP connection	No	C5, C7–C9	C1–C3
R1.6	Near node	TCP connection	No response	C2–C4	-
R1.7	Internet node	TCP connection	Yes	C8–C11	C1–C7
R1.8	Internet node	TCP connection	No	C6, C7	C1–C5
R1.9	Internet node	TCP connection	No response	C1–C5	-
R1.10	Far node	TCP connection	Yes	C11	C1–C8, C10
R1.11	Far node	TCP connection	No	C7	C1–C6, C8–C11

(b) Examples of rules

system that is tailored to crowdsourcing of rules and parallel remote probes.

*1) **Voting-based crowdsourced rules**:* Searching for "network problems" on Google returns millions of web pages. Many of these are linked to Q&A boards where people discuss their symptoms and others suggest possible causes. However, it is very inefficient to visit every site and read every answer to determine a correct solution for their situation. The DYSWIS rule repository is intended to provide a unified platform for collecting such knowledge in a single place. Questions and answers on the Internet are equivalent to the diagnostic rules in DYSWIS. To support crowdsourcing efficiently, we design the rules to be simple and independent. Each rule contains the name of a probing module, a type of node, a probe result, *likely causes*, and *unlikely causes*. *Likely causes* are the causes that the author of the rule believes to be the probable causes when the particular type of node runs the probing module and returns the specified result. On the contrary, *unlikely causes* are the causes that are believed to be irrelevant to the returned result.

When a user creates or updates a rule on the DYSWIS rule website, other experts can judge the new rule and vote; plus one if they think it is true and useful (up-vote), and minus one if it is incorrect (down-vote). The effectiveness of this type of voting has been proven through many crowdsourced social websites such as *Reddit* and *Stackoverflow*. Similar to these websites, the useful rules acquire the attention and greater points. The total voting points for an incorrect or unhelpful rule will be low or even negative.

*2) **Parallel remote probing**:* To diagnose faults, DYSWIS first selects an appropriate set of rules based on the detected symptoms of failures and automatically excludes the rules that have negative or low voting points. Then, it sends probe requests to particular types of remote users according to the rules. Remote users respond with their probing results asynchronously, and whenever a result arrives, the possibility scores of potential causes are updated. The details of the algorithm are described in Algorithm 1.

When a probe result arrives, `ResultReceived` is called. This module finds a rule matched to the received result. It then increases the possibility score of each cause in the *likely causes* list and decreases it in the *unlikely causes* list. For example, if a sister node is asked to run the `TCPConnection` module, it will verify whether a TCP connection to the remote server is successful. If it succeeds, it will respond 'Yes', and we increase the *possibility score* of problem C1 and decrease all the other *possibility scores* according to rule R1.1, as shown in Table IIIb. The results from other collaborative nodes also update the scores, and finally, the cause with the highest score is considered the most probable root cause. After informing the users of the diagnostic results, we can collect useful feedback information from them as to whether the diagnostic result was correct. The statistics obtained from this survey can be used to improve the rules and estimate the occurrence frequencies of the actual causes. In the case where our diagnostic results fail to pinpoint a specific cause, but suggest multiple probable causes, this occurrence frequency will be helpful to infer the one most likely occur among them.

Algorithm 1 Parallel distributed probing

```
1: function PROBE(failure)
2:     Update rule sets from the repository.
3:     for each rule in the rule set for failure do
4:         if rule.nodeType == remote then
5:             Send probing request to a remote node.
6:         else
7:             Run the probing module in a new thread
8:
9: //Invoked when each probing result arrives:
10: function RESULTRECEIVED(result)
11:     R ← FindRuleSet(result.ruleId)
12:     rules ← FindRules(R, result.nodeType, result.response)
13:     for each rule in rules do
14:         if rule.votingPoint>0 then
15:             for each cause in rule.likelyCauses do
16:                 P[cause] ← P[cause] + 1
17:             for each cause in rule.unlikelyCauses do
18:                 P[cause] ← P[cause] − 1
19:     DescendingSort(P)
20:     Update current top possible causes to the users.
```

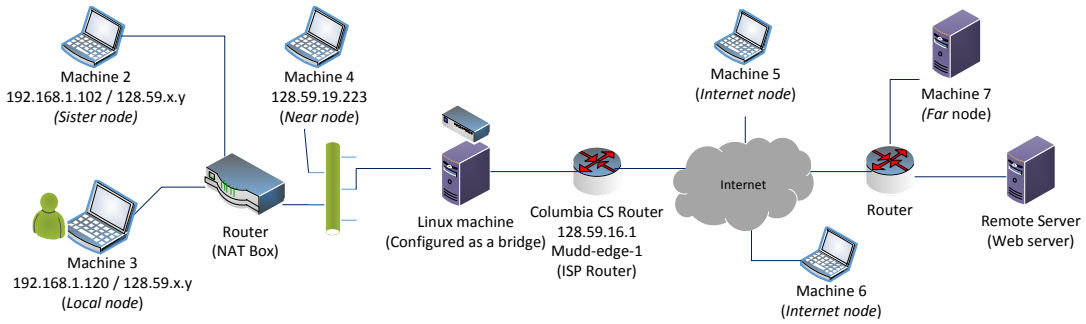


Figure 4: A fault diagnosis testbed for injected failure scenarios

This approach makes crowdsourced rule development and network diagnosis feasible – the independence of rules enables multiple participants to create their rules easily without disturbing other rules. The voting feature enables DYSWIS to exclude useless or incorrect rules, and distinguish more commonly occurring causes. In addition, the separation of rules makes parallel remote probing possible. Since the diagnostic process is not affected by the order of received probe results, a node can distribute probe requests to multiple nodes and process returned results asynchronously, which is faster than sequential probes. Furthermore, this approach can avoid the situation that the entire diagnostic process is misdirected by a few incorrect probes from malicious nodes.

V. Evaluation

A. Experimental testbed

In order to evaluate the capability of DYSWIS, we set up a testbed that contained a NAT box, a bridge, remote servers, and collaborative nodes. As illustrated in Figure 4, we placed *near nodes* inside the campus network, *Internet nodes* in various external networks, and a web server and *far nodes* on the Amazon EC2 network. We simulated Internet service provider (ISP) network failures by injecting network delays or dropping packets on the bridge between the NAT box and the campus network.

B. Common network failures

We compared the diagnostic accuracy of DYSWIS with four diagnostic tools, two tools provided by operating systems (Windows 7 and Mac OS X) and two commercial tools (Network Magic Pro 5.5 from Cisco Systems [6] and HomeNet Manager 3.0.8 from SingleClick systems [7]). We ran each tool in the testbed with injected faults and evaluated the diagnostic result. The failure scenarios were adopted from other studies [8], [9], which investigated common network failures obtained from surveys on end-user environments. We merged them and inserted several additional scenarios to create our test list (Table IV). In this table, ○ implies that the diagnostic result is correct, △ indicates that the result is helpful but imprecise, and × denotes that the tool

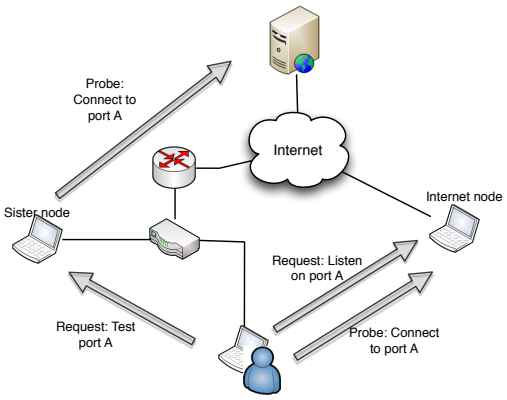


Figure 5: Diagnosis example: Port blocking test

has no capability to diagnose the fault or that it outputs an incorrect answer. The first five failures listed in the table were caused by misconfigurations, and the last five were due to a service outage or port blocking. Although the commercial tools provided many powerful functions such as network monitoring and convenient user interface for network settings, they exhibited limited capabilities in diagnosing our fault scenarios. The tools embedded in each OS also failed to diagnose most scenarios as described in Table IV. The tools performed better in the scenarios of misconfiguration faults; however, they failed to correctly diagnose the outage and port blocking scenarios. This is not surprising since there is no good way to investigate the network infrastructure (NAT, ISP, or remote server) for tools running on end-user machines.

In contrast, DYSWIS successfully identified the root causes in seven out of ten scenarios taking advantage of the assistance of other nodes located in different networks. For example, the blocking of a website by the ISP could be diagnosed by comparing probe results from multiple *near* and *Internet nodes*. If every *near node* failed to connect to a particular server while the *Internet nodes* could connect to the server, we inferred that the traffic between the server and the ISP was constrained.

Similarly, we diagnosed the port blocking problems,

Table IV: The diagnosis results of each problem diagnostic tool for injected fault scenarios.

No.	Injected faults	Windows 7 diagnostic tool	Mac OS X diagnostic tool	Network Magic Pro	HomeNet Manager	DYSWIS
1	Ethernet cable unplugged	O	O	O	O	O
2	Network adapter disabled	O	O	O	O	O
3	IP address conflicts	O	O	X	X	△ (Invalid IP address)
4	Incorrect gateway address	△ (DHCP is not enabled for wireless network connection)	△ (reboot the router)	X	X	△ (Your local gateway router is down or refusing your request)
5	DNS address misconfigured	O	X	X	X	O (You don't have a proper DNS server. Other nodes do not observe the problem on their DNS servers. We recommend an alternative DNS server: xx.xx)
6	Server down (Web or SSH server)	△ (Not accept the connection)	X	X	X	O (Nobody can connect to the server at present. It is very likely that the server is not working at present.)
7	A NAT blocks a server	△ (Not accept the connection)	X	X	X	O (It is very likely that your router blocks the server.)
8	An ISP blocks a server	△ (Not accept the connection)	X	X	X	O (It is very likely that your ISP blocks the server.)
8	Port blocking by NAT (e.g., SSH and BitTorrent)	X	X	X	X	O (The port X is blocked by your router.)
9	Port blocking by ISP	X	X	X	X	O (The port X is blocked by your ISP.)
10	A web server is too slow	X	X	△	△	△ (Pinpoint possible congestion points with additional steps.)

which are common in home networks. If a home router blocks a particular inbound or outbound port, applications that use the port will not function properly. In the problems presented in NetPrints [9], five out of 25 recent home network faults were related to this issue. To diagnose these problems, NetPrints used current configurations on home routers and nodes. Although this attempt can pinpoint misconfigured settings, it is difficult to identify the root cause when packets are blocked by an ISP or remote servers, which usually do not expose their policies. Figure 5 describes the approach of DYSWIS for this issue. By comparing probe results from sister, near, and Internet nodes, we determined whether a particular outbound port was blocked by a local router or an ISP. Further, by asking other nodes to send packets to the local machine via a specific port and comparing the results from different types of nodes, we could determine whether the user needed to reconfigure the router or consult the ISP about the port issue.

Another advantage of this collaboration is that we can obtain alternative solutions. For example, if a local DNS does not function properly, we can temporally configure other DNS servers recommended by external nodes until the local server is recovered. If the outside DNS servers refuse queries from the node because of a security concern, we can also request the collaborating nodes to query the domain to their DNS servers and resolve the IP address on behalf of the *local node*. However, there is a security issue that malicious nodes might provide compromised information. To mitigate this risk, DYSWIS asks multiple nodes to collect multiple alternative solutions and provide the most frequently answered solutions to the users because it is very rare for random collaborative nodes to provide the same compromised information.

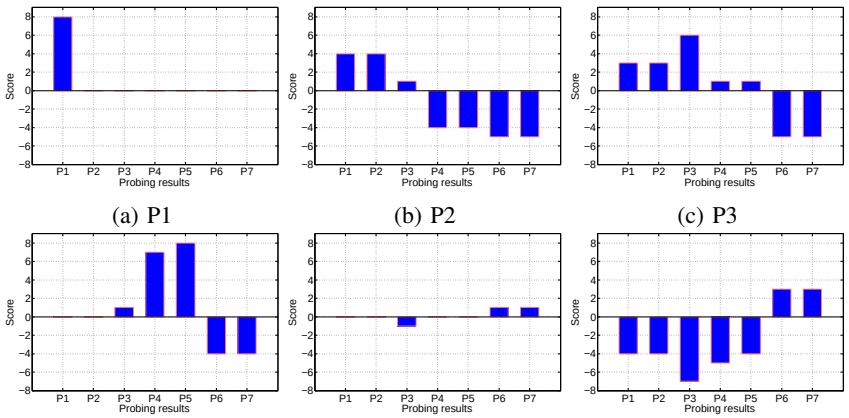
C. Detecting performance bottlenecks

In this section, we describe the detailed diagnostic results of problem #10 ("A web server is slow") listed in Table IV. This kind of performance problem is challenging to diagnose since there are a number of possible points where bottlenecks may be located. We assumed that there were seven candidate congestion points on the path from the client to the remote server. Then, our project members wrote multiple rules independently as described in Section IV. For example, if round-trip time (RTT) between the local node to the web server is very high while RTT between a sister node and the server is considerably lower, we increase the score of `P1` in Table V and decrease the other scores. In a similar manner, we wrote 19 other rules for this case. The full list of the rules can be found on our website[2], and the list can be edited (crowdsourced) by any participant. To evaluate the accuracy, we artificially generated bottlenecks by configuring the packet delay on each device or link. Table V describes the possible bottleneck points and each bar graph in Figure 6 shows the results of DYSWIS obtained from each experiment with injected bottlenecks. The bars indicate the final scores obtained after running the rules. The cause that gained the highest score is the most probable cause. In six out of seven scenarios, the actual point where the delay was injected gained the highest score, which

[2]http://dyswis.cs.columbia.edu/webrules

ID	Fault scenarios
P1	Network adapter disabled
P2	Problems on the link between the user and the router
P3	Problems on the router
P4	Problems on the ISP
P5	Problems on the link between the ISP and the Internet
P6	Problems on the service provider network
P7	Problems on the remote web server

Table V: Possible bottlenecks of the network



(a) P1 (b) P2 (c) P3

(d) P4 and P5 (e) P6 (f) P7

Figure 6: Probing results of each scenario

implies that DYSWIS can pinpoint the bottleneck point correctly. However, in three cases (`P2`, `P6`, and `P7`), there exist two tied entries that gained the same scores. The addition of more rules is helpful to narrow down the root causes of these cases. For example, Figure 6(e) shows that `P6` and `P7` gained the highest score, which implies that DYSWIS could not determine whether the high latency of the web server was caused by the provider network (e.g., Amazon EC2) or by the remote server. In this case, we can request a far node, located in the same provider network, to measure RTT to the target server. If the RTT is high, we can infer that the service provider network may have a problem. Therefore, we can add the following rule – if the RTT from a far node to the target server is high, increase the score of `P6`. By adding this rule, we ensure that `P6` will gain a higher score and DYSWIS can finally narrow down the actual cause appropriately. This process can be repeated and applied to other scenarios by crowdsourcing. We believe that the larger number of rules that are aggregated, the higher will be the system accuracy.

VI. Discussion

Because our approach employs the collaboration among peers, it is susceptible to security issues found in peer-to-peer (P2P) networks [10], which are vulnerable to malicious users who try to attack others by providing malformed data (e.g., file poisoning) or by using manipulated identities (e.g., Sybil attack). Furthermore, in P2P systems, a user's IP address is exposed to others. This makes it easy for malicious users to target a user through denial of service attack, in addition to the privacy issues that such exposure entails. In this section, we discuss the potential security problems in our approach and suggest several solutions.

A. Security issues

Because DYSWIS's network protocol and APIs are open to the public, other applications can participate in the DYSWIS network. However, a disadvantage exists in that a user could contact DYSWIS nodes in order to initiate malicious probes against a normal service. There are two attack scenarios.

The first scenario is a denial-of-service (DoS) attack: A malicious node can simply send a large number of probe requests to a target node, which will then become busy handling these probes. This attack can be prevented by counting the number of requests from other nodes and simply restricts the maximum number of probing requests per node within a particular period. The second scenario is the distributed denial-of-service (DDoS) attack. A malicious user can utilize multiple DYSWIS nodes to launch a DDoS attack by asking them to probe the same node or web server. For example, the malicious user requests multiple users to execute a "TCP connection check" to a target IP address. Because the peers are not aware that these requests are being sent to multiple users by the malicious user, they will execute the requests as usual - open a TCP connection to the target - in a manner similar to how compromised nodes in a typical botnet behave.

In order to prevent this attack, every node that is requested to perform a probe looks up the probe history to check whether the host or service has been probed recently and a usable result exists. This will prevent redundant probes from being performed. However, for this to be effective, every probe transaction performed by each node should be stored. This is not recommended because the database (DHT, in our system) can be flooded with probing transactions. DYSWIS reduces the history size by randomly storing only a small portion (e.g., 10%) of the entire transactions in the DHT because an estimated number of probes is enough for our mechanism. For example, if ten recent probes are detected by querying the DHT, it implies that around 100 probes have been performed recently. If the number exceeds a certain limit, DYSWIS considers it to be a part of an attack and refuses to perform the probe. In this case, the malicious user cannot harm the target, but a normal user can still obtain

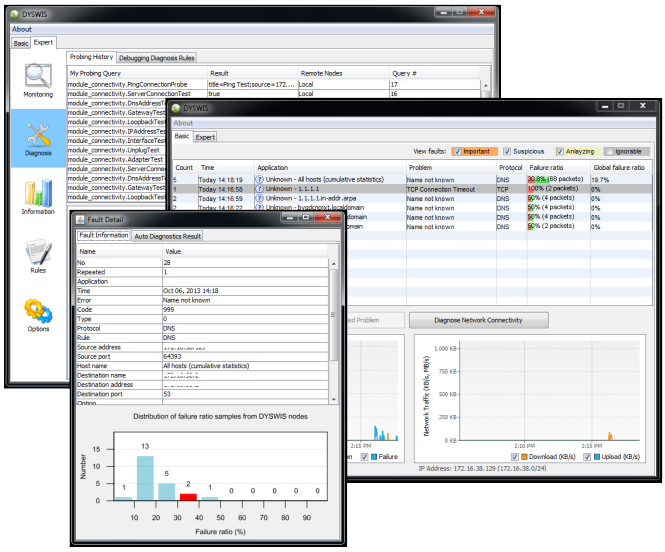


Figure 7: DYSWIS screen dumps

probe results from other nodes.

B. Social Network Peers

Another challenging problem exists, namely, that of whether we can trust the probe results from other nodes. This is because a user might be malicious and could therefore be providing wrong results. In this section, we suggest a mechanism to distinguish genuine users from potentially malicious nodes by recommending *social peers* registered on the friend list in a social network service (e.g., Facebook). This approach is based on the actual human social interactions. When someone needs the answer to an important question, they first ask their friends before asking, say, some random, anonymous person on the street, because they trust their friends more. Similarly, we assume that if we choose collaborative peers among close friends in the social network service, the probability of obtaining trustable peers is much higher than the case of simply obtaining random nodes in a DHT. Thus, if DYSWIS discovers peers who are on the friend list of the user, it recommends those social peers to the user. The user can finally determine whether to choose random or social peers. Because DYSWIS requires only a couple of nodes for fault diagnostics, we do not need to collect hundreds of peers as does a typical P2P file sharing system. In our current diagnosis rules, six nodes are even sufficient to run a diagnosis; this number is reasonably small and this many nodes could easily be retrieved from a user's friend list in a social network. One of the challenges of this system is determining who a user's close friends are. DYSWIS calculates the proximity scores of each friend in Facebook by using the number of wall posts and messages that they have exchanged. This algorithm is heuristic, yet it adequately distinguishes actual friends from fake ones.

However, we also need to consider the privacy issue. It is possible that the IP address of a particular friend could be exposed in this approach. Our goal is to provide the contact points (IP addresses) of social peers without revealing the matches that indicate who has a specific IP address.

We have implemented this system using the Facebook API as a proof of concept and integrated it into the DYSWIS framework. It first generates an identification key for each node. The identification key is a unique MD5 hash string generated from a Facebook user ID and a secret key of the application. Because the secret key is not exposed to the public, it is impossible to reproduce the identification key using the names of friends. Thus, only the user and the Facebook application know the secret key. After receiving the identification key from the Facebook application, DYSWIS registers the (key, IP address) pair in the DHT. When another user requests a social peer list, the Facebook application calculates the proximity of the requested user and returns the closest friends from the Facebook friend list of the user. For this purpose, it is necessary for the user to pass the authentication process beforehand. Note that the application does not return the names of friends or user IDs. Instead, it returns the hash strings that were generated with a user ID along with a secret key. In the last step, DYSWIS queries the DHT to check whether the received keys are registered. In other words, it checks whether the friends have installed DYSWIS and are running it at present. Consequently, through these steps, DYSWIS can obtain the IP addresses of close friends who are running DYSWIS without revealing the actual owner of the IP addresses.

VII. Related Work

Network fault detection and diagnosis have been an area of interest for a number of years. A number of studies discuss home network environments. For example, HomeNet Profiler [11] measured several characteristics of home networks such as the quality of home Wi-Fi networks and the deployment of auto-configuration protocols. Cui et al. [12] identified the root cause of high web page loading time by capturing packets and correlating measured metrics, and Sundaresan et al. [13] determined whether a performance problem was located at the user's ISP or the home network. Also, several studies use the collaboration of different machines to diagnose problems. WebProfiler [14] aggregated observations of multiple machines to discover network elements involved in failures, Netprints [9] diagnosed and resolved problems in home router configurations using shared knowledge of labeled (good or bad) configurations collected from multiple machines, and WiFiProfiler [15] relies on cooperation among wireless clients to diagnose and resolve problems. The main difference of DYSWIS's approach from these studies is that DYSWIS not only uses the failure history of others, but also leverages end-users' active probing in real-time while others rely on passive

observations from the users. By combining the passive and active probings, we filter out false positive failures and diagnose the filtered problems more accurately. Furthermore, in our best knowledge, DYSWIS is the first platform that suggests a practical method that supports a crowdsourced rule repository for network problem diagnosis.

There are several proposals that use user-based diagnosis. For example, Glasnost [16] discovers service differentiation by ISP based on traffic analysis between an end point and another controlled end point in the network. Choffnes et al. [17] proposed a methodology to detect network events based on users' experiences. They aimed to detect events impacting user-perceived application performance. Zhang et al. [18] proposed end user based collaborative active probing to diagnose significant routing events. Tulip [19] probed routers to localize anomalies such as packet reordering and loss. Dasu [20] developed a platform that enables network researchers to experiment network-related issues using a huge number of end users. These studies focus more on investigating the network core elements while DYSWIS focuses end-user problem diagnosis.

AutoMON [21] uses a P2P-based solution to test network performance and reliability. The distributed testing and monitoring nodes are coordinated by using a DHT, which helps in locating resources or agents. This study focuses on testing and monitoring while DYSWIS is designed to diagnose the root cause of failures.

VIII. Conclusion

DYSWIS diagnoses complex end-user's network problems using end-user collaboration. We provide a new framework for collaborative approach and diagnosis strategies for various fault scenarios. We provide a detailed design to discover and communicate with collaborating nodes. Also, we provide a framework for administrators and developers to participate to contribute to expand the diagnostic system.

We have implemented a prototype of the DYSWIS framework and present how easily the participants add new rules and modules on top of the framework in order to diagnose several common network faults. We set up these scenarios with real network devices and diagnosed them using those rules and modules we have created. While local probing with traditional diagnosis tools fail to point out the cause of these fault scenarios, our evaluation presents that DYSWIS can effectively narrow down the problematic regions and pinpoint the root causes.

References

[1] B. Pourebrahimi, K. Bertels, and S. Vassiliadis, "A survey of peer-to-peer networks," in *Proc. of ProRisc*, Veldhoven, The Netherlands, Nov. 2005.

[2] Apple inc, "Bonjour," http://www.apple.com/support/bonjour/.

[3] E. Bozdag, A. Mesbah, and A. van Deursen, "A Comparison of Push and Pull Techniques for AJAX," in *Proc. of WSE 2007*, Paris, France, Oct. 2007.

[4] H. Nam, B. H. Kim, D. Calin, and H. Schulzrinne, "Mobile Video is Inefficient: A Traffic Analysis," in *Proc. of IEEE CTEMD Workshop*, Atlanta, GA, USA, Dec. 2013.

[5] A. Amirante, S. P. Romano, K. H. Kim, and H. Schulzrinne, "Online non-intrusive diagnosis of one-way RTP faults in VoIP networks using cooperation," in *Proc. of IPTComm '10*, Munich, Germany, Oct. 2010.

[6] "Homenet manager," http://www.homenetmanager.com/.

[7] "Network magic pro," http://tinyurl.com/n6hh7ka.

[8] C. Dong and N. Dulay, "Argumentation-based fault diagnosis for home networks," in *Proc. of HomeNets*, Toronto, Ontario, Canada, Aug. 2011.

[9] B. Aggarwal, R. Bhagwan, T. Das, S. Eswaran, V. N. Padmanabhan, and G. M. Voelker, "Netprints: diagnosing home network misconfigurations using shared knowledge," in *Proc. of NSDI*, Berkeley, CA, USA, 2009.

[10] G. Urdaneta, G. Pierre, and M. V. Steen, "A survey of DHT security techniques," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 8:1–8:49, Feb. 2011.

[11] L. DiCioccio, R. Teixeira, and C. Rosenberg, "Characterizing Home Networks With HomeNet Profiler," in *Technical Report CP-PRL-2011-09-0001, Technicolor*, Sep. 2011.

[12] H. Cui and E. Biersack, "Trouble shooting interactive Web sessions in a home environment," in *Proc. of HomeNets*, Toronto, Ontario, Canada, Aug. 2011.

[13] S. Sundaresan, Y. Grunenberger, N. Feamster, D. Papagiannaki, D. Levin, and R. Teixeira, "WTF? Locating Performance Problems in Home Networks," in *SCS Technical Report GT-CS-13-03*, Jun. 2013.

[14] S. Agarwal, N. Liogkas, P. Mohan, and V. N. Padmanabhan, "WebProfiler: cooperative diagnosis of Web failures," in *Proc. of COMSNETS*, Bangalore, India, January 2010.

[15] R. Ch, V. N. Padmanabhan, and M. Zhang, "WiFiProfiler: Cooperative diagnosis in wireless LANs," in *Proc. of MobiSys*, Uppsala, Sweden, June 2006.

[16] M. Dischinger, M. Marcon, S. Guha, P. K. Gummadi, R. Mahajan, and S. Saroiu, "Glasnost: Enabling end users to detect traffic differentiation." in *Proc. of NSDI*, San Jose, CA, USA, April 2010.

[17] D. R. Choffnes, F. E. Bustamante, and Z. Ge, "Crowdsourcing service-level network event monitoring." in *Proc. of ACM SIGCOMM*, New Delhi, India, September 2010.

[18] Y. Zhang, Z. M. Mao, and M. Zhang, "Effective diagnosis of routing disruptions from end systems." in *Proc. of NSDI*, San Francisco, CA, USA, April 2008.

[19] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "User-level internet path diagnosis," in *Proc. of ACM SOSP*, New York, NY, USA, October 2003.

[20] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger, "Dasu: pushing experiments to the internet's edge," in *Proc. of NSDI*, Lombard, IL, Apr. 2013.

[21] A. Binzenhöfer, K. Tutschku, B. auf dem Graben, M. Fiedler, and P. Arlos, "A p2p-based framework for distributed network management," in *Proc. of EuroNGI Workshop*, Villa Vigoni, Italy, July 2005.