MoT: A Collaborative Network Troubleshooting Platform for the Internet of Things

Kyung-Hwa Kim*, Hyunwoo Nam†, Jin-Hyung Park* and Henning Schulzrinne*
*Department of Computer Science, Columbia University, New York, NY
†Department of Electrical Engineering, Columbia University, New York, NY

Abstract—**Troubleshooting network problems on networked home devices is not easy, because most devices have insufficient computing power to run sophisticated diagnostic tools and have no user interfaces to debug the problem directly. We propose MoT, a network problem diagnosis platform that leverages the collaboration of smart objects, smartphones, and computers. We take advantage that recent devices have multiple communication interfaces. Therefore, when a device has a problem with an interface, it can send a probe request to other devices using an alternative interface. We use collaborative mechanisms to diagnose the root cause of a network problem. It can use cooperation from internal nodes or send requests to external nodes. We demonstrate the feasibility of this approach by implementing an Android application and an algorithm that diagnoses a push notification failure.**

I. Introduction

Today, not only smartphones and laptop computers but also traditional household devices such as TVs, air conditioners, lamps, and door locks are networked (smart objects). Although the Internet grants powerful functionality to these smart objects, the convenience instantly turns into a nuisance when the network does not function properly and the cause remains hidden. Troubleshooting network problems on such devices is not easy because most devices have insufficient computing power to run sophisticated diagnostic tools and have no user interfaces to debug the problem directly. According to Sundaresan et al. [1], service calls to Internet Service Providers (ISPs) for network troubleshooting are costly ($9–$25 per call). If the Internet of Things (IoT) environment further penetrates the home and every household device is connected to the Internet, this cost will increase drastically in the near future.

For general end-user computers, the diagnostic tools for network problems can be useful to mitigate the pain of the troubleshooting process. Existing tools include traditional command line tools (e.g., `ping` and `traceroute`), network diagnostic software embedded in each operating system, and several third-party diagnostic tools [2], [3]. However, these tools are not only difficult to use for technically non-savvy users, but also inappropriate for home devices because they require user interfaces such as keyboards and monitors. Moreover, some tools may execute arduous tasks such as packet sniffing to trace and analyze the network packets [4], which requires more memory, storage, and CPU power than the small devices usually possess. For example, it is impractical to connect a monitor and keyboard physically to a networked door lock and execute `ping`, `nslookup`, and `tcpdump` in order to identify the cause of its network problem. Therefore, it is necessary to build a lightweight and user-friendly network diagnostic tool for home networks.

We propose MoT ("Medic of Things"), a network problem diagnosis platform that leverages the collaboration of smart objects, smartphones, and computers. The main idea is that when a device suffers from network problems such as DNS resolution errors, misbehavior of a Network Address Translation (NAT) box, port blocking, and DHCP problems, it offloads the troubleshooting task to other devices that have more capabilities (e.g., network accessibility or computing power). However, there is an issue of how to inform the other devices that the problem has occurred if the network is faulty. We note that most recent smart objects (e.g., smart TVs) have been designed to support extra communication protocols such as Bluetooth, ZigBee, or NFC (Near Field Communication) in addition to Wi-Fi. Therefore, even if a device has a problem with a Wi-Fi network, the problem can be reported to nearby devices via other available communication interfaces. When another device receives the problem report, and that device has diagnostic functionality, it can start the diagnostic process to examine the problematic device and the network. Otherwise, it can simply forward the task to a more suitable device. For example, if a networked refrigerator has a problem with a Wi-Fi network, it sends a diagnosis request (like a distress message) to a nearby laptop computer that has no network problem. The laptop has enough memory, CPU, and user interfaces to run a dedicated diagnostic tool and thus is better able to diagnose the current network status.

We use a common message format as the means through which heterogeneous devices communicate with each other to send, forward, and receive reports of network problems. Moreover, a device that diagnoses a problem can send probe requests to other devices to obtain different information about the current network state from multiple devices. To separate the modules for rules and probing, we designed two layers: *logic layer* and *probe layer*. This enables some devices that have less computing power to have only the probe functionality and others that have sufficient computing power (e.g., laptop computers and tablets) to have both diagnostic and probe functionality. We implemented a prototype of this platform, including applications for Android devices and general computers. The Android application uses native APIs to implement the *probe layer* and adopt a Web application technology for the *logic layer*. In order to obtain probe results from other nodes located outside the home network, the software for the general computers was developed on the basis of DYSWIS ("Do You See What I See?") [5], which is an end-user network diagnostic tool based on a peer-to-peer (P2P) network. We also suggest using instances in public clouds if a user worries about privacy in using P2P nodes. To prove the feasibility, we wrote a sample rule for diagnosis of push notification faults on Android devices.

In Section III, we describe the architecture of MoT, and in Section IV, we discuss several practical scenarios of problem diagnosis. Then, we present the details of the implementation in Section V, and demonstrate the feasibility of this approach in Section VI. Finally, we state our conclusions in Section VII.

II. Related work

A number of researchers have studied home network troubleshooting. Sundaresan et al. [1] diagnosed home networks with poor performance by determining the bottleneck points. Cui et al. [6] designed a Web session troubleshooting tool for home environments by using the correlation between packet measurements made on multiple machines. Aggarwal et al. [7] developed a network diagnostic tool that uses a signature-based learning technique, and Dong et al. [8] wrote an argumentation-based algorithm for home network diagnosis.

We have also developed network diagnostic systems, DYSWIS [5] and WiSlow [9]. These tools were also focused on diagnosing network problems and performance degradation for end users. DYSWIS uses distributed nodes to pinpoint the root cause of a network problem, and WiSlow analyzes 802.11 packets to identify the sources of Wi-Fi performance degradation, such as channel contention and interference caused by non-Wi-Fi devices.

However, not many studies have been focused on networked home devices. Since these devices are different from general computers in terms of user interface and computing power, existing studies are unlikely to be applicable for these devices.

In this paper, we adopt the rule system of DYSWIS and its collaborative method, and suggest additional mechanisms to support home devices. Wustner et al. [10] suggested a similar idea as MoT in terms of collaboration of home network devices for troubleshooting network problems. When a user complains about the low performance of a network, the authors suggested to correlate the different recorded metrics such as RTT, jitter, throughput, and packet retransmission. They determine which metric is related to the poor performance. Our approach shares this idea of cooperation between devices; however, we use real-time probing with predefined diagnostic rules instead of correlating metrics. Thus, our approach does not require devices to record network states. Furthermore, we suggest practical mechanisms such as discovering, forwarding, and probing that support the collaboration of devices effectively.

III. MoT architecture

The main goal of MoT is to diagnose network problems of devices in home networks. The diagnostic process is conducted using the collaboration of nodes in a home network and, if needed, MoT connects to other collaborators in external networks to request probes for problem diagnosis. We use a straightforward communication protocol that enables the devices to exchange problem profiles and diagnostic results with each other.

A. Device types

We define three roles in the MoT system: a *client device* is a smart object that has a network problem, a *forwarding device* passes probe requests from the *client* device to a *diagnostic device*, and a *diagnostic device* actually helps to diagnose the problem.

Each device in a home network has one or more roles, depending on its computing power (e.g., memory and CPU) and its attached user interfaces. For example, a laptop computer, which has ample computing power and user interfaces such as a mouse and a monitor screen, will be a *diagnostic device* because it is suitable to run diagnostic software. The laptop may also be a *client device* since it can request a diagnosis to other devices when it has a network problem. Most other devices have roles as both *client devices* and *forwarding devices*. The forwarding is necessary when a *client device* is not physically close enough to the *diagnostic device* to communicate via Bluetooth or ZigBee.

B. Device registration

Each device first uses a service discovery technology (e.g., Bonjour) to discover a directory server and registers its attributes with the server when they are connected to the network. These attributes include which network interfaces it has (e.g., Wi-Fi, Bluetooth, ZigBee, or 3G/LTE), whether it can be mobile (e.g., smartphones and tablets), and whether it has a capability to run diagnostic software. A *diagnostic device*, which is usually a desktop or laptop, runs a MoT server that maintains a device directory based on these profiles and uses this when it diagnoses a problem. For example, if a 3G network is required to diagnose a particular problem, it first looks up which devices can connect to a 3G network (e.g., smartphones), and if such a device is currently reachable, it sends the probe request to the device via intermediate network available (e.g., Wi-Fi or Bluetooth). We describe this scenario in Section IV-D.

C. Problem description and forwarding

When a device detects a network problem, it first creates a problem description that contains failure symptoms (e.g., it cannot connect to the wireless access point (AP), or the TCP latency to servers in the Internet is too long). Then, it sends a diagnosis request that contains the problem description to the *diagnostic device*. The problem description includes the following parameters.

```
problem description := (deviceId, problemId,
timestamps, problematic interface type, MAC
address, application name, protocol, port,
problem symptoms)
```

However, when a device is located too far from a *diagnostic device*, it cannot send the diagnosis request to the *diagnostic device* via Bluetooth or ZigBee. In this case, our suggestion is that the diagnosis request can be forwarded (broadcasted) to other devices that are in the vicinity of the faulty device. The devices that receive the diagnosis request are responsible for sending it to the *diagnostic device* or forwarding it to other devices that have a connection to the *diagnostic device*.

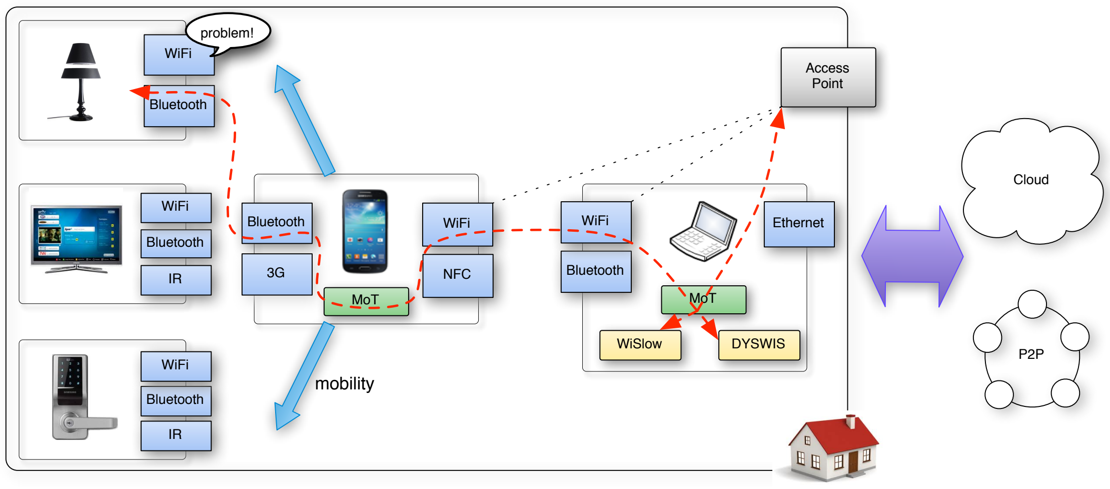


Fig. 1: The architecture of MoT

D. Mobile devices

When a device has a problem with the Wi-Fi network and has no nearby forwarding device, it cannot send the diagnosis request to another device. In this case, mobile devices such as smartphones and tablets have important roles. We take advantage of its portability in order to collect diagnosis requests from problematic devices. The devices that are large or fixed, such as refrigerators, lights, and door locks, cannot be moved and often are located far from other devices (e.g., lights at the ceiling). Therefore, when these devices have problems with the Wi-Fi network, the devices may fail to reach others because of their limited communication range, even if they have active communication interfaces other than Wi-Fi. In this case, a mobile device can be a forwarding device or a diagnostic device. A user carries the mobile device close to the problematic device and pairs via Bluetooth[1]. Then, the problematic device notices that a forwarding (or diagnostic) device, which is portable, is nearby and sends it the diagnosis request.

E. Diagnosis

The diagnostic processes are driven by predefined diagnostic rules. We adopt the rule system from DYSWIS, so the rules can be crowdsourced and updated via a central rule repository server. The basic rule starts with a check of whether the same problem has been reported by other devices. If there has been no report of the problem, we attempt to communicate with other devices to determine whether those are reachable without any network problems. If it turns out that the same problem occurs for multiple devices in the network, we infer that the problem is caused by the home network infrastructure (e.g., an AP). In this case, we run other diagnostic software such as DYSWIS and WiSlow to check whether DNS, DHCP, TCP/UDP, and Wi-Fi function properly. If other devices do not observe the problem, then MoT interacts with the problematic device again via an alternative communication interface and requests that the probe modules indicated by the diagnostic rule be executed. We define a simple message syntax for this request. The message contains the name of the probing module that should be executed and the parameters that should be passed to the module. The return value will be sent back in a similar format. As an example, JSON-format messages are illustrated below.

```
request = {“module”: “TCP listen”, “parameters”: {“port: 80”}}
response = {“status”: “success”}

request = {“module”: “ping”, “parameters”: {“host”: “192.168.1.1”}}
response = {“status”: “success”, “result”:“5ms”}
```

These messages are used between different types of devices in order to exchange probe requests and responses. We open the rule system to any network experts and manufacturers of smart objects.

When a device fails to connect to a server outside the home network (e.g., a web service or IoT management server[2]), it is necessary to obtain probe results from external collaborative nodes. For example, if a device fails to connect to the central management server, it is useful to know whether other devices or computers in different networks (other households) have the same problem. Accordingly, we can use the P2P network that DYSWIS provides, which originally was designed to help general computer users with problem diagnosis. DYSWIS supports a distributed network composed of multiple peer nodes that voluntarily participate in a fault diagnosis process. The nodes run the probes requested to diagnose the network problems of end-users. In a similar way, MoT ask other nodes in the DYSWIS network in order to determine whether the management server is working properly for those nodes. Moreover, collaborative nodes in outside networks can send network packets to the home devices in order to confirm that incoming packets are received correctly.

However, a user may well be concerned about the privacy of using P2P networks, because information on the problem

[1]Currently, the Bluetooth paring process is done manually in our prototype.

[2]For example, devices need to connect to a SECE [11] server, which is a central management server for networked devices. SECE is a general IoT platform developed at Columbia University.

is revealed to other users when probes are requested. If a user prefers a secure method to diagnose the problem, the alternative method that we suggest is the use of virtual instances in public clouds. Since public clouds now offer instances in multiple geographical locations (e.g., Amazon EC2), it is even possible to run probe processes in multiple different networks without the help of a P2P network or a distributed shared network such as PlanetLab [12]. Figure 1 illustrates the collaboration of cloud instances to diagnose home network devices. When a suspicious network behavior is observed, we launch instances from a prepared image that contains probing modules to assist in the problem diagnosis.

IV. Diagnosis scenarios

In this section, we describe several sample scenarios of problem diagnosis process using MoT.

A. Device diagnostics using history

Suppose that the bandwidth of a device is capped by a firewall at an AP for security reasons. Because of this, the device has difficulty connecting to the network and sends a diagnosis request to a nearby laptop via Bluetooth. Then, the laptop starts the diagnostic process by sending probe requests to other devices to determine whether those have the same problem. Then, MoT compares the problem description by the problematic device with probe results from other devices to identify the cause of the problem. In this case, since other devices observe no network problem, MoT can infer that the AP is functioning correctly, but only this device suffers from the bandwidth problem, which implies there is a configuration issue at the AP.

B. Device diagnostics using active probing

Suppose that a device suffers from performance degradation due to severe Wi-Fi interference. The Wi-Fi interference can be caused by channel contention or nearby non-Wi-Fi devices [9], [13]. Such interference degrades the throughput to almost zero when a strongly interfering device such as a baby monitor is located very close to the afflicted device. In this case, the problematic device may even fail to report the problem to a diagnostic node.Therefore, our approach is that a mobile device with a forwarding capability (e.g., smartphone) collects the problem description from the problematic device using an alternative communication protocol such as Bluetooth and then forwards the message to the diagnostic node when the nodes move into areas where the Wi-Fi network is accessible with no problem. Then, the diagnostic node runs specialized diagnostic software such as WiSlow [9] to detect the root cause of the Wi-Fi interference.

C. Smartphone diagnostics

Another example is the diagnosis of a problem with a smartphone. Suppose that a smartphone application has a network problem but the source of the problem is not known. For example, many Android applications use push notifications to send network packets to mobile devices. To test whether these notifications work correctly, the diagnostic application on a smartphone sends probe requests to a laptop via Bluetooth or any other intermediate communication methods available. The laptop diagnoses the problem using predefined diagnostic rules. These diagnostic rules entail further probes into the smartphone via Bluetooth to proceed with the diagnosis. We tested this scenario with an Android application that we implemented (Section V and VI).

D. Computer diagnostics

Another possible scenario is the reverse of the previous scenario. A laptop computer has a network problem, and thus it cannot contact external nodes in other networks. For example, suppose that an ISP has a temporary outage and its customers have no Internet connection. In this case, the laptop becomes completely isolated and there are not many methods to diagnose the causes. With MoT, the laptop first detects that there is a device (smartphone) connected to a 3G network. Then, the laptop sends a diagnosis request to the smartphone which connects to the Internet via the 3G network and diagnoses the problem. Although the 3G and Wi-Fi network use different ISPs and network paths, the 3G network can still assist the diagnosis. For example, the smartphone can obtain real-time service interruption records from the websites that maintain the list of services that are down[3]. Moreover, it can send probing requests to external nodes that are connected to the same ISP to ascertain what is actually happening in the Internet and the service provider network.

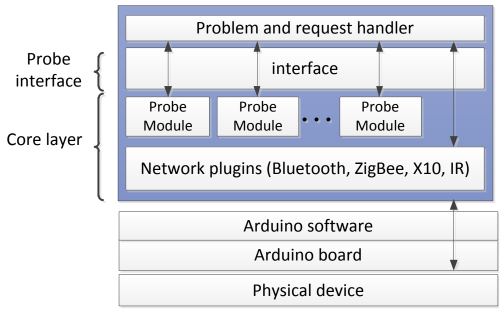
V. Implementation

As a proof of concept, we implemented and tested a prototype of a network troubleshooting tool based on the MoT platform. This implementation includes two applications, one for Android devices and the other for computers. The applications communicate via Bluetooth in this example. Algorithm 1 diagnoses Scenario C in Section IV.

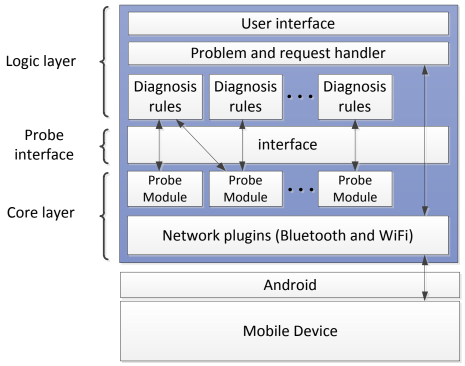
The challenges to implementation are twofold. First, we need to create a framework that is generally applicable to heterogeneous devices. Therefore, it is necessary to have a platform-independent interface. Second, the diagnostic strategies should be changed easily without repackaging software entirely, since the diagnostic logic may be updated frequently. To achieve these goals, we divide the system into two layers (Figure 2): the core layer, which is rarely updated, and the logic layer, which can be updated flexibly. These two layers communicate with each other using the probe messages.

We aggregate platform-dependent modules and place these in the core layer to avoid entirely rebuilding the software when diagnostic algorithms are updated. As a result, probe modules (e.g., `ping`, `traceroute`, and the TCP connection checker) and network communication modules (e.g., for Bluetooth and TCP) are included in the core layer since these functions use the native APIs supported by the underlying platform. Although the modules in the core layer should be developed separately for each type of device, these rarely need to be updated once implemented. The core layer modules that we developed for the Android application include a Bluetooth sender and receiver, a TCP and UDP tester, a DNS tester, and wrappers for traditional tools such as `ping` and `traceroute`.

[3]For example, http://www.downdetector.com



(a) MoT client for Arduino (microcontroller)



(b) MoT client for Android

Fig. 2: Implementation architecture

The logic layer includes diagnostic rules and a user interface. These modules need to be updated frequently, whenever new diagnosis rules are designed. Therefore, a mechanism that dynamically loads and updates new modules is needed. The industry standard for such mechanism is OSGi, which supports dynamic loading of Java classes without entirely recompiling or repackaging software. However, Dalvik VM, which is the Java virtual machine for Android, does not allow dynamic loading of Java classes because each class in Dalvik should be signed at its compile time. As a result, there is no way to insert the class files or JAR files during the runtime of the application.

Therefore, in the MoT client for Android we adopt the hybrid approach, which is popular with mobile application developers because it makes the development cycle faster and updates easier [14]. Thus, we implement the core layer in the native language of Android (i.e., Java), and the logic layer using web applications (i.e., HTML5 and Javascript). This technology is used to separate core network probing modules from diagnostic rules. As a result, we can independently develop and reuse the logic layer without modifying the original application. Thus, by simply writing HTML files and Javascripts, new rules and user interfaces can be added to the application. As described in Section III, the modules within each layer communicate with each other using messages formatted in JSON that contain the names and parameters of probe functions.

Another advantage of dividing the system into two layers is that some devices do not need to have both layers. Although our implementation for Android has both layers, the logic layer is not necessarily installed on small devices that have less computing power and no user interface. Figure 2a describes the prototype of MoT for such devices, which is implemented on top of the Arduino microcontroller [15]. In this model, the device has only the core layer, which communicates with the logic layer of another device to diagnose its own network problem or help the other device with a diagnosis. The messages used between two layers within the same device are also used across different devices when probe requests and responses are exchanged.

VI. Evaluation

The Android application servers send messages to the target Android device using Google's messaging services such as Google Cloud Messaging (GCM) or Cloud to Device Messaging (C2DM). These services enable application developers to transmit push notifications to Android device users [16], [17]. Although a number of Android applications use these services, the users have no good way to diagnose a problem if they suspect that their applications do not receive the notifications properly.

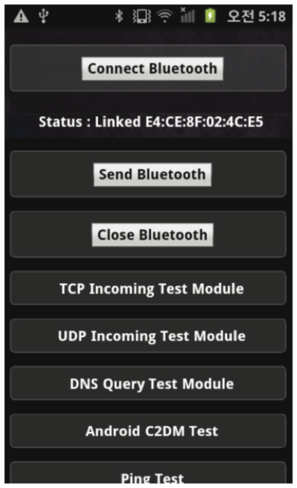
Algorithm 1 C2DM Test

```
1:  function PROBE($failure$)
2:      $D \leftarrow$ the problematic Android device
3:      Request(C2DM Server, send a message to $D$)
4:      if $D$ received C2DM message then
5:          return "A non-network problem"
6:      else
7:          $N_e \leftarrow$ an external node
8:          Request($N_e$ open a TCP port)
9:          Request($D$, send TCP packets to $N_e$)
10:         if $D$ successfully sends packets to $N_e$ then
11:             return "A problem in the GCM or C2DM servers"
12:         else
13:             $N_i \leftarrow$ an internal node
14:             Request($N_i$ open a TCP port)
15:             Request($D$, send TCP packets to $N_i$)
16:             if $D$ successfully sends packets to $N_i$ then
17:                 return "A problem with the connection to the ISP"
18:             else
19:                 return "A problem in the local network"
```

We implemented the Android network diagnostic tool, which can determine whether a mobile device is able to receive a push notification correctly. When the push notification system is not functioning properly, there are four possible cases:

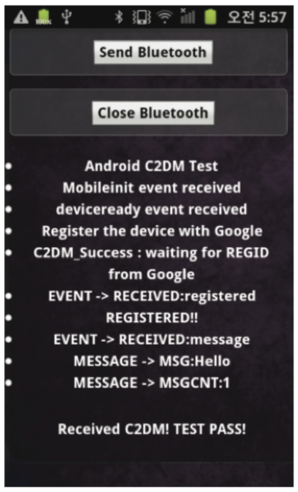
- A problem in the GCM or C2DM servers
- A problem with the connection to the ISP
- A problem in the local network
- A non-network problem

We assume that the device is using a Wi-Fi network. To identify the cause of the network problem, our Android



```
Status : Linked E4:CE:8F:02:4C:E5
```

(a) A GUI of test modules



```
• Android C2DM Test
• Mobileinit event received
• deviceready event received
• Register the device with Google
• C2DM_Success : waiting for REGID
  from Google
• EVENT -> RECEIVED:registered
  REGISTERED!!
• EVENT -> RECEIVED:message
• MESSAGE -> MSG:Hello
• MESSAGE -> MSGCNT:1

Received C2DM! TEST PASS!
```

(b) Android C2DM test

Fig. 3: An MoT application for Android devices

application first sends a diagnosis request to a laptop via Bluetooth, since we cannot be sure that the Wi-Fi network is functioning correctly in this example. On behalf of the Android device, the laptop runs a diagnostic algorithm (Algorithm 1). First, it requests the Google server to send a push notification to the Android device. If the device fails to receive this notification, the second test is performed to determine whether the device can exchange TCP packets with an external node in the Internet. We use P2P nodes or cloud instances for this test as described in Section III. If this succeeds, we can infer that there is no problem in connecting to the Internet from the device. Then, we attempt to send packets to the device from the laptop to determine whether the local area network is faulty. If everything works but the device still cannot receive the push notification, we can infer that the push notification servers are the cause rather than other networks. We tested the tool using our testbed, which artificially injects local network connectivity faults. Also, we simulated the outage of C2DM servers by using our own servers instead of Google servers in our testbed. The tool successfully distinguished the local network problems from the push notification fault in our testbed. Figure 3 shows the screenshots of the tool for Android.

VII. Conclusion

We designed a network troubleshooting system, MoT, which supports the collaboration of home devices (smart objects) and end-user devices such as laptop computers and smartphones. We take advantage that recent devices have multiple communication interfaces. Therefore, when a device has a problem with an interface, it can send a probe request to other devices using an alternative interface. Moreover, we focused on a mobile device that is able to move physically close to other problematic devices and collect problem profiles. Finally, the system adopts collaborative mechanisms to diagnose the root cause of a network problem. It can use cooperation from internal nodes or send requests to external P2P nodes or cloud instances. We demonstrated the feasibility of this approach by implementing an Android application and an algorithm that diagnoses the push notification failure.

References

[1] 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.

[2] “HomeNet Manager,” http://www.homenetmanager.com/, [Online; accessed Dec 2013].

[3] “Network Magic Pro,” http://tinyurl.com/n6hh7ka, [Online; accessed Dec 2013].

[4] “Network diagnostics in Windows 7,” http://tinyurl.com/k3xj8x6, [Online; accessed Sep. 2013].

[5] K. H. Kim, V. Singh, and H. Schulzrinne, “DYSWIS: Collaborative Network Fault Diagnosis - Of End-users, By End-users, For End-users,” in *Columbia University Technical Report cucs-017-11*, 2011.

[6] H. Cui and E. Biersack, “Trouble shooting interactive web sessions in a home environment,” in *Proc. of HomeNets ’11*, Toronto, Ontario, Canada, Aug. 2011.

[7] B. Aggarwal, R. Bhagwan, L. De Carli, V. Padmanabhan, and K. Puttaswamy, “Deja vu: fingerprinting network problems,” in *Proc. of CoNEXT ’11*, Tokyo, Japan, Dec. 2011.

[8] C. Dong and N. Dulay, “Argumentation-based fault diagnosis for home networks,” in *Proc. of HomeNets ’11*, Toronto, Ontario, Canada, Aug. 2011.

[9] K. H. Kim, H. Nam, and H. Schulzrinne, “WiSlow: A Wi-Fi Network Performance Troubleshooting Tool for End Users,” *To appear in IEEE INFOCOM*, Toronto, Canada, April 2014.

[10] S. Wustner, D. Joumblatt, R. Teixeira, and J. Chandrashekar, “Automated home network troubleshooting with device collaboration,” in *Proc. of CoNEXT Student ’12*, Nice, France, Dec. 2012.

[11] O. Boyaci, V. Beltran, and H. Schulzrinne, “Bridging communications and the physical world: Sense Everything, Control Everything,” in *Proc. of GLOBECOM Workshops ’10*, Florida, USA, Dec. 2010.

[12] “PlanetLab,” https://www.planet-lab.org/, [Online; accessed Jan 2014].

[13] S. Rayanchu, A. Patro, and S. Banerjee, “Airshark: Detecting non-WiFi RF Devices Using Commodity WiFi Hardware,” in *Proc. of ACM IMC*, Berlin, Germany, Nov. 2011.

[14] “Hybrid Apps,” http://www.nngroup.com/articles/mobile-native-apps/, [Online; accessed Jan 2014].

[15] “Arduino,” http://www.arduino.cc/, [Online; accessed Sep. 2013].

[16] “GCM,” http://developer.android.com/google/gcm/, [Online; accessed Sep 2013].

[17] “C2DM,” https://developers.google.com/android/c2dm/, [Online; accessed Sep 2013].