Tracking Middleboxes in the Mobile World with TraceboxAndroid

Valentin Thirion, Korian Edeline, and Benoit Donnet(✉)

Université de Liège, Liège, Belgium
{benoit.donnet,korian.edeline}@ulg.ac.be

Abstract. Middleboxes are largely deployed over cellular networks. It is known that they might disrupt network performance, expose users to security issues, and harm protocols deployability. Further, hardly any network measurements tools for smartphones are able to infer middlebox behaviors, specially if one cannot control both ends of a path. In this paper, we present TraceboxAndroid a *proof-of-concept* measurement application for Android mobile devices implementing the `tracebox` algorithm. It aims at diagnosing middlebox-impaired paths by detecting and locating rewriting middleboxes. We analyze a dataset sample to highlight the range of opportunities offered by TraceboxAndroid. We show that TraceboxAndroid can be useful for mobile users as well as for the research community.

1 Introduction

It has been a while, now, the Research Community has tried to use computing resources made available by end-users. One of the most famous tentative is probably SETI@home [1]. SETI@home provides a screensaver that users can freely install, and that downloads and analyzes radio-telescope data for signs of intelligent life. The project obtains a portion of the computing power of the users' computers, and in turn the users are rewarded by the knowledge that they are participating in a collective research effort, by attractive visualizations of the data, and by having their contributions publicly acknowledged. This model has been followed by others, in particular for performing large-scale Internet topology data collection [2,3].

Meanwhile, we have seen the rise of mobile devices at the expense of a drop in desk-based and notebook computers [4]. Mobile data usage is increasing rapidly in part due to a growing release of mobile devices and the availability of a wide variety of mobile phone applications. For instance, a large number of users nowadays use their mobile devices to watch video on demand (e.g., YouTube) while on the move. Naturally, the idea of using crowdsourcing with mobile devices to measure the Internet has emerged. However, the particularities of those devices make network measurements a difficult task [5].

This work is funded by the European Commission funded mPlane ICT-318627 project.

M. Steiner et al. (Eds.): TMA 2015, LNCS 9053, pp. 79–91, 2015.
DOI: 10.1007/978-3-319-17172-2_6

In addition, the Internet infrastructure has strongly evolved. In particular, we have seen the growing importance of *middleboxes* (i.e., "an intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host" [6]). For instance, Sherry et al. [7] obtained configurations from 57 enterprise networks and revealed that they can contain as many middleboxes as routers. Wang et al. [8] surveyed 107 cellular networks and found that 82 of them used NATs. Those middleboxes has shown to have a negative impact on the evolvability of the TCP/IP protocol suite [9].

Combining the rise of mobile devices and the importance of middleboxes leads to a natural question: are the middleboxes a cause of degraded performance in mobile networks and can we provide tools for monitoring and discovering problems with end-user Quality of Experience (QoE) due to their presence. The first part of the question has already been answered. Wang et al. [8] have demonstrated that middleboxes are, indeed, a brake to mobile network performance in mobile networks.

In this paper, we tackle the second part of the question by proposing a proof-of-concept tool called *TraceboxAndroid*. TraceboxAndroid is based on `tracebox` [10], a `traceroute` extension that is able to reveal the presence of middleboxes along a path. We have ported `tracebox` under the Android system, allowing so the user to detect the presence of a middlebox that could be the cause of a degraded performance on a path. We describe the architecture of TraceboxAndroid and deploy it on several mobile devices across the world in order to demonstrate its potentialities. In addition, TraceboxAndroid is lightweight for mobile devices in terms of battery and memory consumption. TraceboxAndroid is freely available (http://androidtracebox.org).

The remainder of this paper is organized as follows: Sec. 2 explains how `tracebox` works; Sec. 3 presents TraceboxAndroid, our `tracebox` port into Android devices; Sec. 4 explains our TraceboxAndroid deployment, data collection, and results we obtained; Sec. 5 positions this paper regarding the state of the art; finally, Sec. 6 concludes this paper by summarizing its main achievements.

2 Tracebox

To reveal the presence of middleboxes along a path, we use `tracebox` [10], an extension to the widely used `traceroute`.

`tracebox` mechanism is illustrated in Fig. 1. It relies on RFC1812 [11] and RFC792 [12] stating that the returned ICMP `time-exceeded` message should quote the IP header of the original packet and respectively the complete payload or the first 64 bits. `tracebox` uses the same incremental approach as `traceroute`, i.e., it sends packets with different IP, UDP, or TCP fields and options with increasing TTL values. By comparing the quoted packet to the original, one can highlight the modifications and the initial TTL value allows us to localize the two or more hops between which the change took place. In Fig. 1, packet `a` is the originally sent one. The first hop, that happens to be a middlebox, modifies

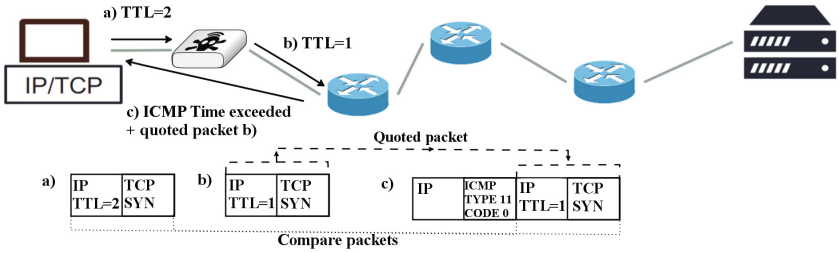


Fig. 1. Example of middlebox detection with tracebox

its TCP Initial Sequence Number (ISN) and sends the rewritten packet `b` to the next hop. When the next hop receives the expired packet, it sends back to the client an ICMP `time-exceeded` packet `c` containing packet `b` as a payload. When the `tracebox` client receives it, it is able to compare packet `a` and the payload of packet `c` to detect any changes and the initial TTL value, i.e., 2, allows `tracebox` to bound the middlebox location.

It is worth to notice that in 80% of the cases [10], a path contains at least one router which implement RFC1812 [11], that recommends to quote the entire IP packet in the returned ICMP. This means that, in most cases, `tracebox` is able to detect any modification performed by upstream middleboxes.

3 TraceboxAndroid

`tracebox` has been originally developed to work on desk-based computers, on a UNIX-like system. We have ported `tracebox` on Android mobile devices. Our application is called *TraceboxAndroid*.

Fig. 2 illustrates the general TraceboxAndroid architecture. As shown, it is made of three main components: the *system core* where the `tracebox` intelligence has been included (coded in C, under the front office), the *front office* (or the *application*) corresponding to the Android application (coded in Java) and the *back office* (or *server* – coded in PHP and HTML) that is used to store data and make offline analysis.

The front office communicates with the server using an XML API that gives the application the destinations to be probed and allows it to send back the data collected by the system core. The core itself implements `tracebox` (as described in Sec. 2) and sends probes to the destinations using sockets by system calls.

The front office and the system core are freely available[1] since mid-2014. In the subsequent sections, we deepen each component.

3.1 System Core

TraceboxAndroid is based on BusyBox [13], a software written in C that aims at providing Unix tools for operating systems with limited resources. In particular, it contains a basic networking tool suite (e.g., `ping`, `netstat`, `netcat`,

[1] http://www.androidtracebox.org

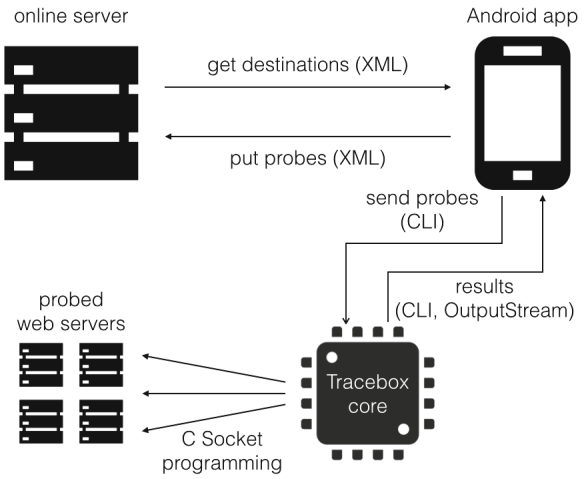


Fig. 2. General overview of the TraceboxAndroid architecture

`traceroute`, etc). We started from the latter source code to port `tracebox` into Android systems.

The main challenge we had to address is that `tracebox` requires the use of raw sockets to manually set IP fields, TCP fields, and TCP options, and retrieve the received headers. As the use of raw sockets is restricted to users that can grant the `CAP_NET_RAW` POSIX capability (i.e., super users), we chose to exclude non-*rooted* devices and call the TraceboxAndroid BusyBox implementation from the JAVA app as a super user. While this limitation is crippling for extended use, as already stated by Faggiani et al. [5], it seems reasonable for a *proof-of-concept* implementation.

TraceboxAndroid *system core* consists in a C applet which is called by a lightweight BusyBox version. The underlying `tracebox` implementation is based on the algorithm described in Sec. 2 and supports TCP/IPv4 and diverse TCP options (e.g., SelectiveACK, Timestamp, MSS, WindowScale, MultipathTCP).

3.2 Front Office

The *front office* (or application) purpose is to send `tracebox` probes to predetermined or user-chosen destinations, compute the result, and send it back to the *back office*. In some sense, it acts as a proxy between the system core and the back office.

On the first execution or when a new version is released, three actions are automatically performed: check if the phone is rooted, download and install our custom lightweight BusyBox version containing the `tracebox` implementation, and retrieve an XML file containing the destination set (detailed in Sec. 4.1).

Three probing mechanisms are available in the Front Office menu:

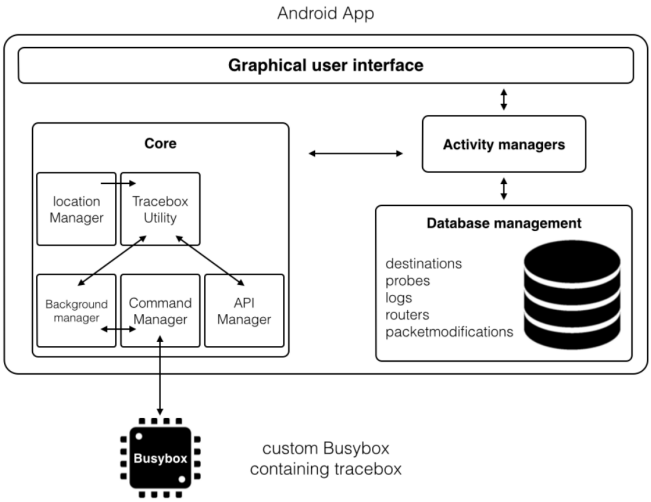


Fig. 3. Front Office organization

- *Background Probing*: background probes are sent by a scheduled task that is executed several times a day while the phone is connected to a network, even if the user shuts down the application.
  The user is able to edit the configuration of the scheduled probing by setting the number of destinations to be probed during one probing session and the maximum duration of a session.
- *Instant Probing*: an Instant probe is a one-time single measurement sent to a random destination within the destination set. The user can choose to run an Instant probe at any time and retrieve the results.
- *Custom Probing*: the Custom probing mode is similar to the Instant Probing but allows the user to set the destination.

Background Probing is mostly for research purposes. The two others are much more dedicated to given network monitoring/analysis in case of a drop in the QoE observed by the user.

The user is also able to check the results, consult the logs, and get information about the application and all the tools that were used to create it.

The architecture of the front office is displayed in Fig. 3. It is divided in three packages; (*i*) the `Main` package that is mainly composed of Activity classes that are responsible for drawing the views, monitoring the state of the app, receiving user commands, and launching `AsyncTask` to perform various operations (e.g., send an Instant Probe, parse an XML file, etc); (*ii*) the `Core` package contains utility and long-lived action managing classes that executes Unix commands and fetch the results, sends the result to the *back office* component and runs the `tracebox` applet; (*iii*) finally, the `Data` package is composed of classes maintaining information about processes and is responsible of managing an internal SQLite database.

The database stores information about destinations, probes, routers, packet modifications, and the logs. In addition to the probed routers inferred characteristics, the GPS position of the device at the probing time, Internet connection mode (i.e., WiFi, cellular, or Bluetooth), the cellular mode and carrier name (in case of a cellular network connection), and the battery consumption are saved into the local database before being sent to the *back office*.

3.3 Back Office

The *back office* is the server-side application that stores data collected by the mobile devices. It also manages the destinations probed by the application.

The *back office* has no other purpose than research (i.e., conserving collected data and off line analysis). The user cannot access the *back office* directly but has the opportunity to analyze data if he selected Instant or Custom probing, or to download the dataset on the website.

4 Evaluation

In this section, we explain our evaluation of TraceboxAndroid. In particular, we discuss our measurement methodology, describe our dataset, and analyze our results.

4.1 Methodology

We built an initial probe target set from the Alexa top-500 websites list, that we pre-resolved using Google Public DNS into 406 unique addresses, avoiding so a resolution on every mobile device that could consume undue resources and that could lead to completely different probed paths, making statistics meaningless. This address set is used by TraceboxAndroid *Background Probing* and *Instant Probing* features. It is, however, obvious that in case of selecting the *Custom Probing* feature, the DNS resolution will be done by the app.

Between May, 2014 and September, 2014 TraceboxAndroid has been downloaded by 23 users from Belgium, Italy, USA, China, and Nigeria. Measurements performed by those users during this period reached a total of 1,756 probes sent. Participating mobile devices were connected to the Internet via WiFI or cellular data networks via different carriers (Mobistar, O2, Mobile Vikings, E-Plus, BASE, T-Mobile, Movistar, KPN) using different mobile technologies (HSPA, HSPAP, HSDPA, LTE, UMTS, and EDGE).

On the whole set of probes sent, 1,372 (78.13%) were done through WiFi connections. The remaining 384 probes (21.87%) were sent through cellular connections.

This dataset is limited but sufficient to demonstrate the extent of TraceboxAndroid capabilities. Moreover, as the Android app was still under development during most measurements, the following results and figures should be considered as illustrations of the variety of the app capabilities rather than rigorous observations.

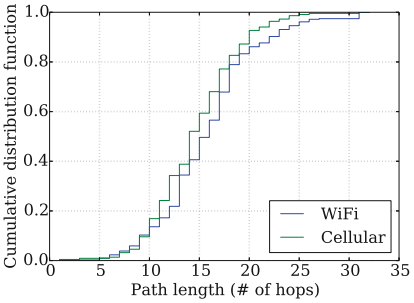


Fig. 4. Path lengths distribution

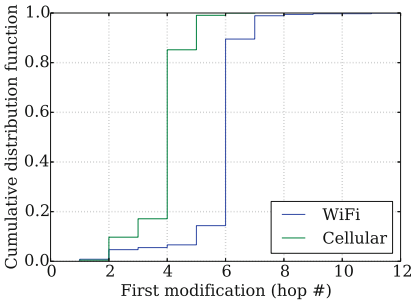


Fig. 5. Location of first observed middlebox modification

4.2 Results

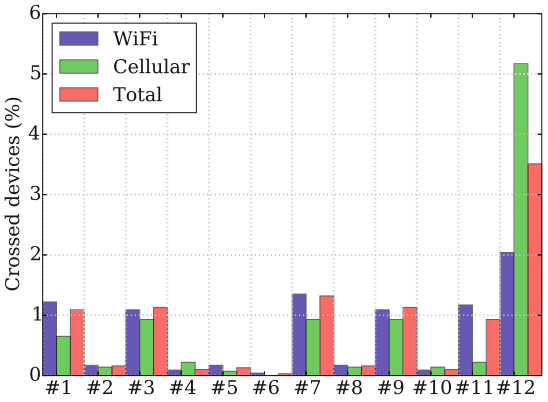
We first look at paths collected and, in particular, at their length distribution. Path lengths are computed on a subset of the paths collected. From the whole set, we select those that have unique <source;destination> address pairs to compute their lengths. This subset contains 606 paths, 388 of them were obtained via WiFi connections (64.9%) and 218 via cellular connections (35.97%).

The results are displayed in Fig. 4. We see that WiFi paths are 1.14 hops longer on average than cellular networks paths, their respective path length means being 15.8 and 14.67 hops.

The location of the first observed middlebox, in number of hops away from the probing source, is shown in Fig. 5. We see that 272 among 361 (75.32%) WiFi paths and 147 among 215 (68.37%) cellular paths that involves a middlebox had their first probe modified close to the mobile device, respectively at hops 6 and 4.

WiFi probes have crossed 180 different different ASs (Autonomous Systems) and cellular probes have crossed 139 different ASs. The AS overlap between the two types of probes includes 111 ASs. The three autonomous systems that WiFi probes have traversed the most are `HIBERNIA TripartZ,NL`, `BELGACOM-SKYNET, BE` and `TTNET,TR`, for cellular probes these are `BASE-AS,BE`, `KPN Interational, NL` and `UUNET,US`. The types of crossed ASs are somewhat equivalent, whatever the type of network connection of the device (i.e., WiFi or cellular), consisting mainly in Transit networks.

We next check IP and TCP modifications. They are inferred using the `tracebox` algorithm described in Sec. 2. As probes have common subpaths, we counted each answering device only once based on the source IP addresses. The resulting set is composed of 3,109 routers, 175 (5.63%) of them exhibit middlebox behaviors. We have observed that 2,304 routers answered through WiFi connections and 1,392 were probed through cellular connections, among them `tracebox` respectively detected 103 (4.47%) and 87 (6.25%) middleboxes. Note that 587 routers have been probed via WiFI and cellular data networks.



(a) Modification rates

Label	*Field*
# 1	IP::ToS
# 2	IP::TotalLength
# 3	IP::ID
# 4	IP::Flags
# 5	IP::Protocol
# 6	IP::Checksum
# 7	TCP::SourcePort
# 8	TCP::DestPort
# 9	TCP::SeqNumber
# 10	TCP::Offset
# 11	TCP::WindowSize
# 12	TCP::Checksum

(b) Legend

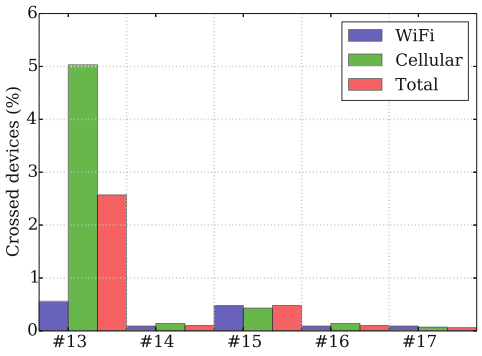
Fig. 6. Observed middlebox modification in IP and TCP headers

The amount of detected middleboxes in this set has to be put in perspective with their strategic positioning; from the unique paths set explained above, 576 among 606 (95.05%) paths are crossing at least one middlebox that modifies at least one IP header, TCP header field, or TCP option. 361 among 389 (93.04%) WiFi paths and 215 among 218 (98.62%) cellular network paths involves a rewriting middlebox.

Fig. 6 summarizes those modifications. The TCP checksum is recomputed by many middleboxes, those that modifies IP pseudo-header fields, TCP fields, and TCP options. It is natural to see that 3.5% of the total observed routers are modifying it. Besides the TCP checksum, four fields are rewritten more often: IP `ToS`, `IP-ID`, TCP source port and TCP sequence number. This modification set exactly matches NATs rewriting behavior.

IP `ToS` rewriting can either come from routers using its DiffServCodePoints (DSCP) sub-field to mark packets for differentiated services, or modifying the last two Explicit Congestion Notification (ECN) bits. The latter modification can either be the action of legacy routers trying to modify the legacy 8-bits ToS field instead of the 6-bits DSCP field, unintentionally modifying ECN-related bits, or a systematic clearance of ECN bits [14].

In several operating systems, `IP-ID` fields of self-forged packets are filled with the value of a globally-incremented packet counter, which is known to be a side-channel leaking information about other connections [15]. Security consequences when endpoints use such a counter to write `IP-ID` have been discussed multiple times and involves enabling attackers to perform idle scan attacks, NATted hosts counting, facilitating TCP injections and more [16–19], but consequences when it is performed by middleboxes for either self-forged or certain non self-forged packets have been less discussed. One of the most harmful known exploitation of middleboxes using a globally-incremented `IP-ID` is when it is combined with TCP window-checking, as it enables attackers to gain feedback on in-window/out-of-window packets to infer the TCP sequence number, and perform off-path TCP injections [18,20,21].



(a) Modification rates

Label	*TCP option*
# 13	TCP::Option_MSS
# 14	TCP::Option_WS
# 15	TCP::Option_Timestamp
# 16	TCP::Option_SACK
# 17	TCP::Option_MPTCP

(b) Legend

Fig. 7. Observed middlebox modification in TCP options

Source port modification is a common practice of Carrier-Grade NATs (CGNs), which makes it difficult for traffic intended for machines located behind it to pass through (e.g., active FTP). However, solutions to this problem have been proposed [22]. TCP ports modification by middleboxes also makes it difficult to achieve transport layer security (e.g., IPSEC) [23].

TCP sequence number modification is mostly due to initial sequence number (ISN) re-shuffling middlebox policies, which aim at mitigating ISN prediction attacks [10]. Such policies are known to create inconsistencies with TCP options using absolute sequence numbers such as Selective ACKnowledgement (SACK), and to reduce substantially the maximum achievable bandwidth [24,25].

TCP Options modifications are shown in Fig. 7. We witnessed multiple middleboxes rewriting the MSS in cellular networks to 1,392 bytes, which is probably designed to obtain packets of 1,500 bytes taking into account the sizes of the desired headers. Among the other options, MultiPathTCP have been cleared by some middleboxes to forbid its use, and is also prone to be blocked by middleboxes that are not familiar with it [25]. WindowScale have been modified by a few middleboxes to custom values, probably for network performance optimization purposes. WindowScale is also known to cause connectivity problems with certain firewalls that do not implement it as defined in RFC1323 [26,27].

Overall, we showed that TraceboxAndroid can be used efficiently for deducing certain network disruption causes from inferred middlebox policies. This can be useful to users for fast on-demand troubleshooting purposes, for researchers that could analyze the collected dataset to get insights such as the permeability of a TCP option, and for network managers to understand what is *really* happening to packets crossing their networks.

Table 1. Observed Memory and CPU consumption

Case	*Samsung Galaxy SII*	Arnova 10d G3
Memory	10.8Mb	6.45Mb
CPU (app)	< 1 %	< 1 %
CPU (instant probe)	12.5 %	12.5%

4.3 Impact on Mobile Devices

To test the impact of the use of TraceboxAndroid on mobile phones, we used the Android Monitoring tool [28]. In more than 99% of the cases, a *Background Probing* session with 10 destinations consumes less than one percent of battery. The same probing session never sent more than 165Kb of data, including the XML result file sent to the *back office*. We also did CPU and memory consumption measurements whose results are shown in Table 1. The experiments were made on a Samsung Galaxy SII (1,4Ghz, 1GB RAM, Android 4.3) and a Arnova 10d G3 (1,2Ghz, 1GB RAM, Android 4.1.1). Clearly, TraceboxAndroid is lightweight for mobile devices.

5 Related Work

Since the end of the nineties, the Internet topology discovery has been extensively studied [29]. In particular, `traceroute` has been used for revealing IP interfaces along the path between a source and a destination. Since then, `traceroute` has been extended in order to mitigate its intrinsic limitations. From simple extensions (i.e., the types of probes sent [30]) to much more developed modifications. For instance, `traceroute` has been improved to face load balancing [31] or the reverse path [32]. Its probing speed and efficiency has also been investigated [33,34].

Medina et al. [24] report one of the first detailed analysis of the interactions between transport protocols and middleboxes. They rely on active probing with `tbit` and contact various web servers to detect whether Explicit Congestion Notification (ECN), IP options, and TCP options can be safely used. The `TCPExposure` software developed by Honda et al. [9] is closest to `tracebox`. It also uses specially crafted packets to test for middlebox interference. Wang et al. [8] analyzed the impact of middleboxes in hundreds of cellular networks. This study revealed various types of packet modifications. More recently, Craven et al. [35] proposed TCP HICCUPS to reveal packet header manipulation to both endpoints of a TCP connection. HICCUPS works by hashing a packet header and by spreading the resulting hash into three fields (in case one is changed). Finally, Xu et al. [36] analyzed the behavior of proxies deployed by four major US cellular carriers. They looked at the HTTP traffic between their clients and their own server. They exhibited mostly application-level proxy features such as

caching, HTTP redirection, image transcoding and connection persistence, and quantified their effectiveness.

These tools provide great results, but they are limited to specific paths as both ends of the path must be under control or must implement particular techniques in the TCP/IP stack and, except for Wang et al. and Xu et al., are not dedicated to mobile devices. On the contrary, TraceboxAndroid does not require any cooperation with the service and only the source must install TraceboxAndroid. It allows one to detect middleboxes on any path, i.e., between a source and any destination.

6 Conclusion

In this paper, we introduced TraceboxAndroid, a `tracebox` port under the Android system, allowing so the user to detect the presence of a middlebox that could be the cause of degraded performance on a path. We showed the extent of TraceboxAndroid capabilities for detection and location of middleboxes rewriting IP and TCP headers fields and TCP options as well as for AS path analysis and `traceroute`-like path displaying.

The main limitation of TraceboxAndroid is the impossibility to forge network and transport headers and read ICMP control messages in non-rooted environments. We need raw sockets to achieve this and their use is restricted to users that can grant the `CAP_NET_RAW` POSIX capability (i.e., super users). As a workaround, we chose to develop a *proof-of-concept* app for rooted devices only and call the TraceboxAndroid BusyBox implementation from the JAVA app as a super user, but this requirement is inappropriate for large-scale deployment because it involves a loss of warranty and risks of system instabilities, among others [37]. Note that this limit has already been discussed in the literature [5].

The dataset sample that we analyze in this paper is fairly limited and does not provide particular insight of middleboxes in mobile networks. However, we believe this dataset is enough to describe the potentialities of TraceboxAndroid.

In the near future, we would like to improve TraceboxAndroid. For instance, we would like to extend the *Custom Probing* mechanism to allow the user to select more IP fields, TCP fields, and TCP options to check and to choose the probe transport layer between TCP and UDP. We also plan to support additional TCP options, such as the TCP Authentication Option (TCP-AO) [38] or the TCP Alternate Checksum Request [39]. Additionally, we would like to improve the user experience by displaying more information and statistics (RTTs, values of modified fields, crossed ASs, etc.) within the application itself.

Another interesting improvement would be to implement middlebox TCP option blocking inference. Mobile devices could send multiple probes with different TCP options combinations to infer middlebox blocking behavior and find if in-path middleboxes are forbidding the use of certain option. This would allow user to perform more complete on-demand connectivity tests and the research community would benefit from the compiled results dataset.

References

1. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@home: An experiment in public-resource computing. Communications of the ACM **45**(11), 56–61 (2002). http://setiathome.berkeley.edu/
2. Shavitt, Y., Shir, E.: DIMES: Let the internet measure itself. ACM SIGCOMM Computer Communication Review **35**(5), 71–74 (2005). http://www.netdimes.org
3. Chen, K., Choffnes, D., Potharaju, R., Chen, Y., Bustamante, F., Pei, D., Zhao, Y.: Where the sidewalk ends: Extending the Internet AS graph using traceroutes from P2P users. In: Proc. ACM SIGCOM CoNEXT, December 2009
4. Rivera, J., Van Der Meulen, R.: Forecast: Devices by operating system and user type, worldwide, 2010–2017. Technical Report 1Q13 Update, Garnter Inc., April 2013. http://www.gartner.com/resId=2396815
5. Faggiani, A., Gregori, E., Lenzini, L., Mainardi, S., Vecchio, A.: On the feasibility of measurement the Internet through smartphone-based crowdsourcing. In: Proc. IEEE International Symposium on Modeling and Optimization in Mobile, Ad-Hoc and Wireless Networks (WiOpt), May 2012
6. Carpenter, B., Brim, S.: Middleboxes: Taxonomy and issues. RFC 3234, Internet Engineering Task Force, February 2002
7. Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., Sekar, V.: Making middleboxes someone else's problem: network processing as a cloud service. In: Proc. ACM SIGCOMM, August 2012
8. Wang, Z., Qian, Z., Xu, Q., Mao, Z., Zhang, M.: An untold story of middleboxes in cellular networks. In: Proc. ACM SIGCOMM, August 2011
9. Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., Tokuda, H.: Is it still possible to extend TCP. In: Proc. ACM/USENIX Internet Measurement Conference (IMC), November 2011
10. Detal, G., Hesmans, B., Bonaventure, O., Vanaubel, Y., Donnet, B.: Revealing middlebox interference with tracebox. In: Proc. ACM/USENIX Internet Measurement Conference (IMC), October 2013
11. Baker, F.: Requirements for IP version. RFC 1812, Internet Engineering Task Force, June 1995
12. Postel, J.: Internet control message protocol. RFC 792, Internet Engineering Task Force, September 1981
13. Vlasenko, D.: BusyBox: the swiss army knife of embedded Linux. http://www.busybox.net
14. Kühlewind, M., Neuner, S., Trammell, B.: On the state of ECN and TCP options on the internet. In: Roughan, M., Chang, R. (eds.) PAM 2013. LNCS, vol. 7799, pp. 135–144. Springer, Heidelberg (2013)
15. Gilad, Y., Herzberg, A.: Spying in the dark: TCP and tor traffic analysis. In: Fischer-Hübner, S., Wright, M. (eds.) PETS 2012. LNCS, vol. 7384, pp. 100–119. Springer, Heidelberg (2012)
16. Bellovin, S.M.: A technique for counting NATed hosts. In: Proc. ACM SIGCOMM Internet Measurement Workshop (IMW), November 2002
17. Zalewski, M.: Silence on the Wire: a Field Guide to Passive Reconnaissance and Indirect Attacks. No Starch Press (2005)
18. Gilad, Y., Herzberg, A.: Off-path attacking the web. In: Proc. 6th USENIX Workshop on Offensive Technologies (WOOT), August 2012
19. West, M., McCann, S.: TCP/IP field behavior. RFC 4413, Internet Engineering Task Force, March 2006

20. Qian, Z., Mao, Z.M.: Off-path TCP sequence number inference aattack - how firewall middleboxes reduce security. In: Proc. IEEE Symposium on Security and Privacy (SP), May 2012
21. Qian, Z., Mao, Z.M., Xie, Y.: Collaborative TCP sequence number inference attack: how to crack sequence number under a second. In: Proc. ACM Conference on Computer and Communications Security (CCS), October 2012
22. Wing, D., Cheshire, S., Boucadair, M., Penno, R.: Port control protocol (PCP). RFC 6887, Internet Engineering Task Force, April 2013
23. Aboba, B., Dixon, W.: IPsec-network address translation (NAT) compatibility requirements. RFC 3715, Internet Engineering Task Force, March 2004
24. Medina, A., Allman, M., Floyd, S.: Measuring interactions between transport protocols and middleboxes. In: Proc. ACM SIGCOMM Internet Measurement Conference (IMC), October 2004
25. Hesmans, B., Duchene, F., Paasch, C., Detal, G., Bonaventure, O.: Are TCP extensions middlebox-proof? In: Proc. Workshop on Hot Topics in Middleboxes and Network Function Virtualization, December 2013
26. Jacobson, V., Braden, R., Borman, D., Satyanarayan, M., Kistler, J.J., Mummert, L.B., Ebling, M.: TCP extension for high performance. RFC 1323, Internet Engineering Task Force, May 1992
27. Microsoft: Network connectivity fails when you try to use Windows Vista behind a firewall device. Technical report, Microsoft (2012). http://support.microsoft.com/kb/934430
28. Android Developers: Device monitor. http://developer.android.com/tools/help/monitor.html
29. Donnet, B., Friedman, T.: Internet topology discovery: a survey. IEEE Communications Surveys and Tutorials **9**(4), December 2007
30. Luckie, M., Hyun, Y., Huffaker, B.: Traceroute probe methode and forward IP path inference. In: ACM SIGCOMM Internet Measurement Conference (IMC), October 2008
31. Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., Teixeira, R.: Avoiding traceroute anomalies with paris traceroute. In: Proc. ACM/USENIX Internet Measurement Conference (IMC), October 2006
32. Katz-Bassett, E., Madhyastha, H., Adhikari, V., Scott, C., Sherry, J., van Wesep, P., Krishnamurthy, A., Anderson, T.: Reverse traceroute. In: Proc. USENIX Symposium on Networked Systems Design and Implementations (NSDI), June 2010
33. Donnet, B., Raoult, P., Friedman, T., Crovella, M.: Efficient algorithms for large-scale topology discovery. In: Proc. ACM SIGMETRICS, June 2005
34. Beverly, R., Berger, A., Xie, G.: Primitives for active Internet topology mapping: Toward high-frequency characterization. In: Proc. ACM/USENIX Internet Measurement Conference (IMC), November 2010
35. Craven, R., Beverly, R., Allman, M.: Middlebox-cooperative TCP for a non end-to-end Internet. In: Proc. ACM SIGCOMM, August 2014
36. Xu, X., Jiang, Y., Flach, T., Katz-Bassett, E., Choffnes, D., Govindan, R.: Investigating transparent web proxies in cellular networks. In: Mirkovic, J., Liu, Y. (eds.) PAM 2015. LNCS, vol. 8995, pp. 262–276. Springer, Heidelberg (2015)
37. Kingo: Warranty disclaimer (2014). http://www.kingoapp.com/root-disclaimer.htm
38. Touch, J., Mankin, A., Bonica, R.: The TCP authentication option. RFC 5925, Internet Engineering Task Force, June 2010
39. Zweig, J., Partridge, C.: TCP alternate checksum options. RFC 1145, Internet Engineering Task Force, February 1990