Measuring The Energy Impact of Security Protocols

Akshay Narayan Michael Chen
CS261N, Vern Paxson
University of California, Berkeley

Abstract

In recent years, mobile devices have become increasingly similar to traditional desktop computers, and the use of common security protocols on such devices is now prevalent. Unlike the traditional computers for which common security protocols were designed, however, mobile devices are constrained by battery life. Unfortunately, the effects of security protocols on mobile device battery life are not well understood, especially on modern mobile operating systems.

In this paper we outline a methodology for measuring the energy cost of security protocols on mobile devices. We further present initial findings from our application of this methodology that TLS can cause a 39% overhead, Tor 36%, VPNs 29%, proximity-based authentication 2%, and mobile antivirus 6%. Based on our findings we argue that the network security community should incorporate energy concerns into the design of mobile security mechanisms.

1 Introduction

Recent years have brought about a revolution in mobile devices. Whereas in the past mobile devices were categorized as embedded systems, today mobile devices are split into two categories: the Internet of Things (IoT) and mobile computing. The Internet of Things is the spiritual successor to the embedded systems space, inheriting the same concerns regarding limited compute power, minimal operating systems, and constrained battery life.

Meanwhile, as Figure 1 demonstrates, mobile computers use the Internet in ways similar to traditional personal computers but are still constrained by battery life.[1] As a result, security protocols (such as TLS [1] and Tor [2]) on mobile computing devices are identical to those used in traditional computers. Unfortunately, such protocols were built from a functional perspective and thus not designed with energy constraints in mind. Therefore, the use of such protocols creates an energy cost that has heretofore been under-appreciated in network security literature.

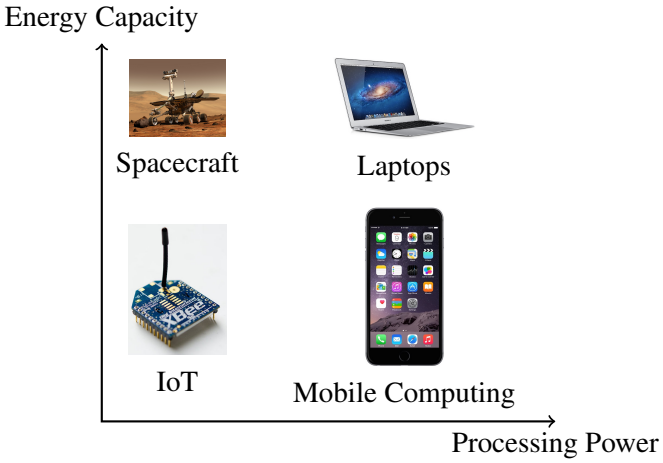

Figure 1: Overview of the battery-powered device space. We focus on the lower right quadrant.

Users who observe this cost may conclude that the benefits of using security protocols on mobile devices are not worth the increased energy use. Therefore, the energy cost of security could be an obstacle to the widespread deployment of security measures. We therefore argue that the tradeoffs between battery life and the use of security protocols should be better understood. Importantly, we do not perform an exhaustive analysis of the effects of security measures on battery life, but rather seek to understand whether the issue is significant.

To further this goal, we make two contributions:

(1) We outline our methodology for measuring the energy cost of a security protocol on a mobile device. We measure energy costs from a user-level perspective, which breaks from the methods used in prior work.

(2) We present our initial findings from applying

[1] Laptop computers are also constrained by battery life, but not nearly to the same extent.

our methodology to measure the energy cost of commonly used security measures. We measured TLS, Tor, VPNs, mobile antivirus applications, and proximity based device authentication. To our knowledge, no prior analysis of the energy costs of these protocols on modern mobile devices exists.

The rest of this paper is organized as follows. Section 2 discusses prior work in the energy analysis with respect to security. Section 3 introduces our methodology and evaluates its merits. Section 4 describes our findings from evaluating common security protocols. Section 5 summarizes our findings and suggests directions for future work.

2 Related Work

2.1 Wireless Sensor Networks

The initial work investigating the energy impact of security came from the wireless sensor networks (WSN) community, the spiritual predecessor of the Internet of Things movement. WSN devices are small, have low-power processors and limited battery life [3]. WSN security literature focused [3, 4, 5, 6, 7] on designing security protocols in a resource constrained environment and did not focus specifically on the energy costs of security.

Nevertheless, some wireless sensor network security literature is relevant to our interests. Wander *et al.* [8] found that using elliptic-curve cryptography could make public-key cryptography viable from an energy perpective even on constrained CPUs. Fonseca *et al.* [9] developed a precise yet lightweight instrumentation system to measure energy usage in embedded systems by modifying device drivers to track changes in component energy states. Finally, Karagiannis *et al.* [10] compared the results of estimating WSN energy usage using simulations to the measured power draw of embedded devices; they found that the results of simulations deviate slightly from measured power usage.

2.1.1 Energy-Based Denial of Service Attacks

Further work pertinent to energy concerns in WSNs and the Internet of Things involved what Stajano and Anderson [11] dubbed *sleep deprivation torture attacks*, a type of denial of service attack in which an attacker forces a device to remain in a high-power state. Martin *et al.* [12] found that this style of attack could reduce battery life in PDAs by a factor of 30 to 280. Brownfield *et al.* [13] developed a link-layer protocol that resists sleep deprivation attacks, increasing WSN node lifetime when under attack from 6 days under 802.11 to 478 days. Racic *et al.* [14] found that a vulnerability in MMS delivery allowed attacks to drain mobile phone batteries 22 times faster.

2.2 Mobile Computing

2.2.1 Power Management

To help maximize battery life in mobile devices, Mittal *et al.* [15] developed a tool to help application developers estimate the energy consumption of their applications, noting that poorly written applications can cause a 30 percent overhead in battery use. Jinal *et al.* [16] and Vekris *et al.* [17] sought to use a runtime system and use static analysis to verify the absence of locks that prevent the device from entering a low-power state. Ding *et al.* [18] studied the impact of wireless signal strength on battery usage and found that a poor WiFi signal cost 810.5% more energy, while a poor 3G signal cost 52% more energy. Finally, Caroll and Heiser [19] measured a smartphone's power consumption by measuring the voltage drop across the smartphone's components and found that the screen and wireless radios were the most energy-intensive.

2.2.2 Mobile Malware

Malware on mobile devices has been a concern since before the advent of modern smartphones and tablets [20, 21, 22, 23, 24]. Kim *et al.* [25] suggested that device power monitoring could identify malware by generating a "power signature" of known malware and comparing it to running applications, but Hoffmann *et al.* [26] argued that high variance in per-process power consumption limits this approach. This result suggests that measuring the energy overhead at the device process level would yield noisy results.

2.2.3 TLS

Potlapally *et al.* [27] were the first to specifically study the energy use of SSL/TLS and found that the SSL handshake dominated connection energy usage for short connections, while symmetric cipher operations dominate energy usage for longer connections. This study predates the advent of modern smartphones. Moreover, we discuss in section 3 that energy measurements by measuring a device's

power draw from a connected power cable are too noisy.

3 Methodology

Measuring battery use is inherently tricky due to the number of factors involved in power usage. In our methodology, instead of trying to decouple the systems involved in power usage, we hope to analyze the end-to-end energy impact of each security protocol. Therefore, we developed an Android application that uses the common security protocols HTTPS, Tor, and VPNs and measures the resulting battery drain using the Android API [28].

3.1 Android Application

We implemented our application using version 21 of the Android API in 928 lines of Java code.

Each measurement utilizes the `AsyncTask` library in Android, which executes tasks from a pool of background threads.[2] In each `AsyncTask`, we run the test suite of choice in a loop to emulate possible user activity and reduce the effect of noise on our measurements. To log battery usage, we register a `BroadcastReceiver` with the operating system to receive notifications for changes in battery level. In order to reliably log at any point in time, we request a WakeLock from the Android OS; this allows the application to prevent the CPU from entering a low-power state.

3.2 Accuracy of Battery Measurement

The battery measurement API in Android returns measurements in integer precision representing the percentage of battery remaining. This measurement has two limitations:

- **Battery Health** The amount of energy a given battery can store degrades over time. Therefore, one percentage point of battery is not comparable across devices, since different devices have experienced different battery usage patterns. Therefore, when running tests on different physical devices, we established separate baseline measurements so that our comparisons remained accurate.

[2] Android does not allow networking in a primary thread, as the OS dedicates these threads to user interface to avoid lag.

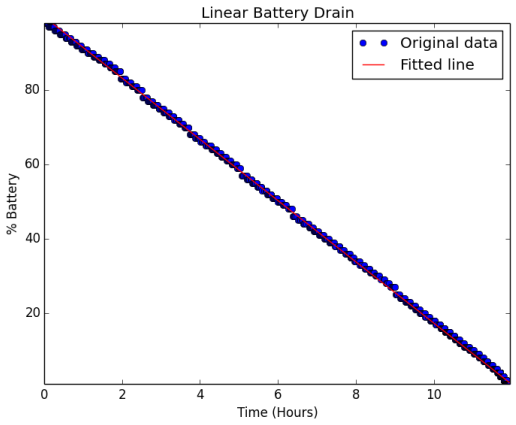

Figure 2: As the battery drains, battery readings return results that decrease linearly. $r^2 = 0.9996$

- **Precision** The amount of time a typical network flow is active is smaller than the amount of time taken to drain one percent of battery life.[3] Therefore, in order to measure the energy cost of a protocol, we must measure the overall energy drain of multiple serial connections. Unfortunately, this means our measurements must be in aggregate, and we cannot analyze per-connection data to determine, for example, the variance of energy use between connections. This variance could be significant due to ambient temperature, wireless signal strength, and packet loss rate.

Notwithstanding these concerns, we have determined that our measurement is an accurate measure of remaining battery life. We demonstrate this by idling the device and measuring remaining battery life at regular intervals. As shown in Figure 2, the observed battery drain is linear. This means that the loss of a certain percentage of battery life corresponds to the same amount of energy use as the same percentage of battery life at a different time.

3.2.1 Measurements Using a Power Cable

We attempted to measure energy use using the method outlined by Potlapally *et al.* [27], but found the results too noisy to be of value. We used a Charger Doctor CPKT002 to measure our device's draw on its power cable while the device idled, idled with its screen on, and performed HTTP GET requests with the screen on. We measured the current

[3] This is not true for long-lived connections and on devices with low battery capacity.

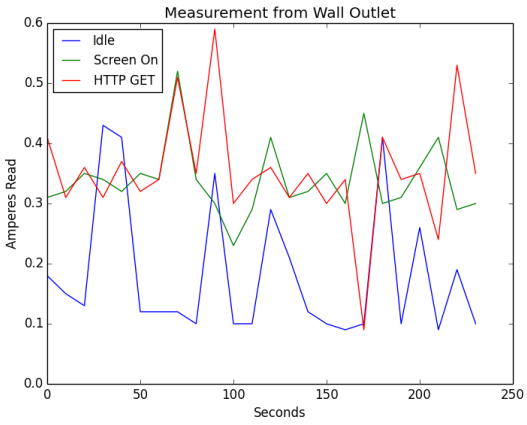

Figure 3: Readings over time of current draw on the power cable.

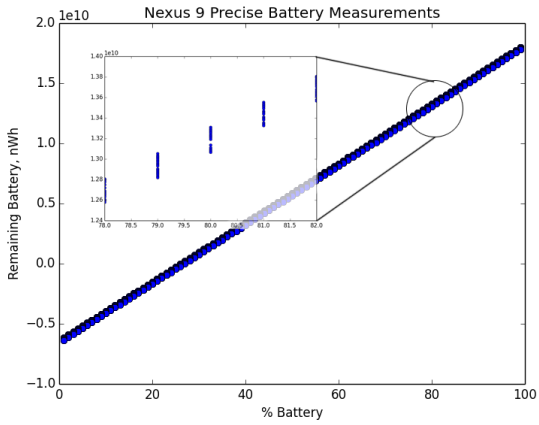

Figure 4: Scatter plot of the precise measurements on our newer Nexus 9 hardware.

drawn every 10 seconds over a period of 250 seconds for each state. As seen in Figure 3, while there is a difference between idling with screen on and with screen off, any energy overhead from performing HTTP GET requests was lost to noise.

3.2.2 More Precise Measurements on Newer Hardware

Power measurements can be more precise with newer hardware and software (in our case a Nexus 9 running Android Lollipop). The Android API specification [28] for accessing precise energy readings states that the kernel measures the battery to a resolution of 8nAh at a nominal voltage of 3.7 volts. Figure 4 shows that readings demonstrate the same linear trend seen in the less precise measurements given by the Nexus 7. The inner graph shows how much more precise the new measurement is — for every integer drop in percentage, there are on average 15 finer grained measurements from the precise API.

As seen on the left hand side of the graph, the kernel reports inaccurate negative readings for precise battery measurements.[4] As we therefore cannot rely on the accuracy of the instrumentation, we do not present further results from our precise instrumentation hardware and leave the precise measurement of individual connections' battery cost to future work.

More broadly, the goal of analyzing the power efficiency of these security protocols is to represent what a user would realistically observe when using these protocols. Therefore, while precise logging gives insight to the low-level fluctuations and dynamic range in connection power usage, our high-level measurement of user energy usage using imprecise APIs remain valid.

3.3 Assessing Energy Usage

In order to assess the power usage of security protocols, we first establish a baseline for each test which performs the same high-level task. For example, we use HTTP as a baseline for HTTPS. Because different security protocols target different high-level tasks, we establish a different baseline for each task. Establishing a baseline allows us to control for variables which affect battery drain so that the only independent variable is the use of a given security protocol.

We use the following tests when gathering results.

- **Direct Web Connections** Issues a GET request for a certain website and reads the website into memory. This simulates a typical network-enabled application on Android.
- **Page loads** Rather than issue the GET request manually, we issue an `Intent` to the operating system. Android resolves this `Intent` to the default web browser (Google Chrome in our case). This test simulates a user in a mobile browser. We recorded the top 70 sites in

[4] Negative ampere-hours are illogical for batteries, so this could be a mistake in the API documentation for Nexus device battery readings.

Alexa's [29] ranking to approximate real user browsing activity.

- **Downloads** Our application issues `Intents` for the operating system to download files from specified URLs. We designed this test suite to test antivirus suites that scan downloads.

- **Idle** A passive security mechanism (*e.g.,* antivirus) runs in the background while the device idles. The radios in the device are allowed to drop to a low power state to emulate a user not actively interacting with the device.

We execute these test suites in loops where we set the iteration count to tradeoff between experiment duration and overall battery drain. Too few iterations results in negligible battery drain, while too many iterations causes experiments to take a long time and limits the number of experiments we can run. While experiments execute, the Android OS notifies the application of changes in battery level at regular intervals. The application logs these battery updates in persistent local storage.

3.4 Decoupling Dimensions

As described in section 4, we found that some security mechanisms cause a latency overhead. It is difficult to separate the causes of the increased energy use that results — did the experiment use more energy simply because it took more time and caused the device to use more CPU cycles, or did the experiment require the use of higher power energy states?

We attempted to create a test that conducts the same operations as the direct web connections test but constrains the task to 3 seconds. If a connection finished early, the CPU WakeLock remained active but we allowed the thread to sleep. If the connection did not complete in time, the application moved on to the next iteration. This test would compare only energy use and not duration.

Unfortunately, we experienced difficulty implementing this test in Android. First, we attempted to set timeouts within our `AsyncTask` and use `Thread.sleep()` if a test finished early, but we found that Android did not always return control of the CPU after the specified sleep period.

We then created a `TimerTask` to periodically launch our `AsyncTasks`, but Android ran these tasks with some variance from the scheduled time, which did not suit our needs for strict 3 second windows.

We then used ScheduledExecutorService to schedule launching our `AsyncTask` at a fixed rate, but we found that calling the `execute()` method on an `AsyncTask` only put our task into a pool of threads used for `AsyncTasks` in general, and Android could schedule our task for a later time.

We also looked at the AlarmManager API, but this API does not provide an interface for setting a precise repeating alarm.[5]

Overall, because Android requires forces network operations to use background threads scheduled by the operating system, we did not find an effective way to force tasks to take a fixed amount of time.

4 Evaluation

After designing a methodology to conduct experiments, we tested TLS, Tor, VPNs, mobile antivirus software, and smartwatch authentication to analyze their power efficiency. For each protocol, we picked an appropriate test based on the user-level activity the protocol secures. Throughout our results we observed high variance between data points; more experiments would give more insight into this variance. Our findings are therefore preliminary.

4.1 Hardware

We performed our experiments using two 2012 Nexus 7 Android tablets with Nvidia Tegra 3 SoC, 1 GB of RAM, and 4326 mAh batteries.

4.2 Server

To study the effect of page size and to help control for server load, we deployed a custom server to the web hosting service Heroku. The server supports returning responses of arbitrary size over both HTTP and HTTPS. We implemented the server in 37 lines of Python code.

4.3 TLS

We used HTTPS connections to measure TLS energy use and used HTTP as a reference baseline.

[5] AlarmManager provides a function `setRepeating()`, but as of API version 19 this is an inexact alarm.

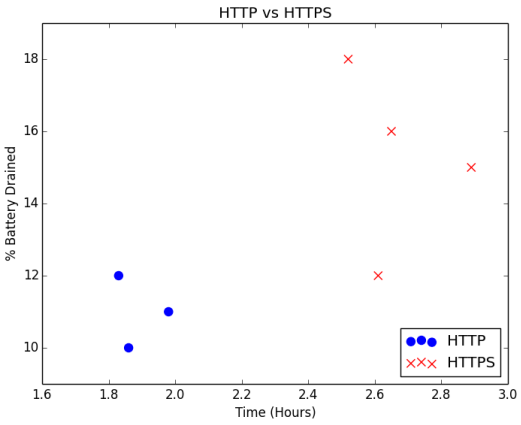

Figure 5: Energy usage and time connecting to `eecs.berkeley.edu`

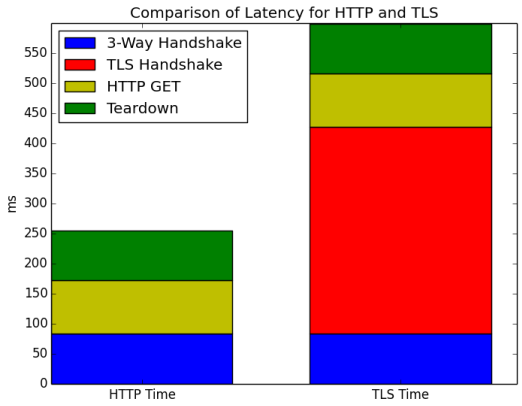

Figure 7: Time used in each step to execute HTTP GET for a 1 KB page from our Heroku server.

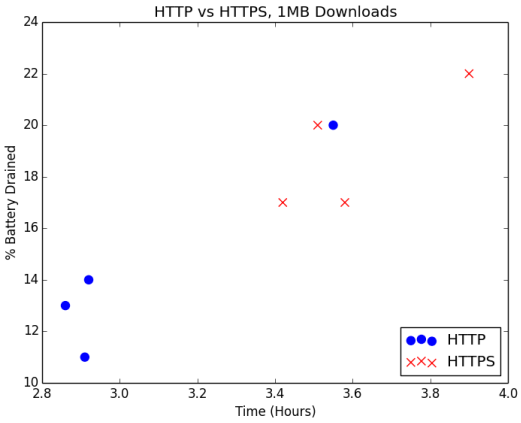

Figure 6: Energy usage and time for 1 MB downloads from Heroku

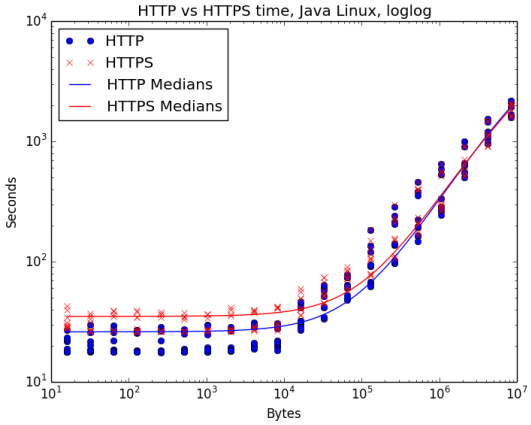

Figure 8: Runtimes for HTTP and HTTPS with varying page size.

We ran our direct web connections test with 50,000 iterations and accessed `eecs.berkeley.edu`, which supports both HTTP and HTTPS. We chose this website both because it is hosted in Berkeley (which reduced variances in network latency) and because unlike other websites we investigated, the HTTP version of `eecs.berkeley.edu` does not redirect users to use HTTPS. The two versions of the site are within 2 bytes of each other in size and are identical other than markings identifying the site as the HTTP or HTTPS version. As shown in Figure 5, on average, HTTPS presents a 40% time overhead and a 39% energy overhead over HTTP.

We also used our direct web connections test on our Heroku server. We again used 50,000 iterations, but fixed the page size at 1 MB. As seen in Figure 6, in this scenario HTTPS has a 25% time overhead and a 50% power overhead compared to the `eecs.berkeley.edu` experiment.

4.3.1 Decoupling Time

We notice that TLS incurred a significant time overhead, but that the time overhead decreased when the page size increased. We hypothesized that the time overhead may be due to the TLS handshake. We tested our hypothesis in two ways.

(1) We collected a Wireshark [30] trace of a single HTTP and HTTPS connection to our Heroku server for a 1 KB page and a 1 MB page. We then divided the trace into connection phases and compared the time each protocol spent

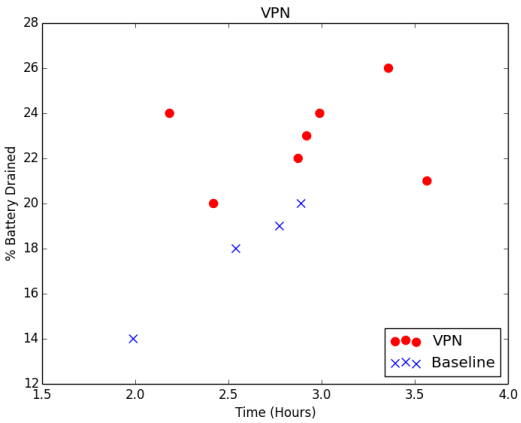

Figure 9: Energy usage and time of VPN test

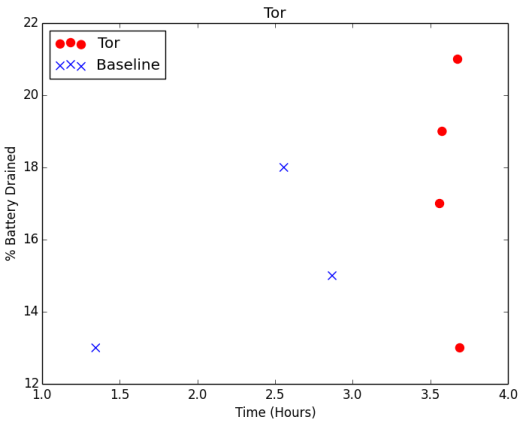

Figure 10: Energy usage and time of Tor test

in each phase. As Figure 7 shows, the only difference between HTTP and HTTPS is the TLS handshake time. Furthermore, this time remains constant as page size increases, so with the 1 MB connections the latency overhead decreases.

(2) We ran HTTP and HTTPS connections from Java on a desktop machine, which uses the same TLS library as Android. We made requests to our Heroku server for page sizes ranging from 16 bytes to 16 MB. As observed in Figure 8, we observe an overhead for TLS when page sizes are small, but this overhead diminishes as page sizes increase.

Both measurements confirm that the TLS handshake causes a significant time overhead. This in turn causes an energy overhead because the wireless radio and screen must remain in high-power states for longer periods of time.

4.4 VPN

We analyzed VPN with our direct web connections test. To set up the VPN, we connected through the UC Berkeley Library VPN using the Cisco Anyconnect application. This app redirected all network traffic from the device through the VPN. We also noted that running the additional VPN app could incur some energy overhead, but this would also emulate the local energy overhead of connecting to a VPN. As seen in Figure 9, using a VPN connection on average takes 14% longer and uses 29% more battery.

4.5 Tor

We analyzed Tor using our direct web connections test. We used Orbot, the official Tor client for Android which if installed on a rooted device will send all network traffic on the device through Tor. As seen in Figure 10, sending traffic through Tor on average takes 96% longer and uses 36% more energy. The dramatic increase in duration is due to the nature of Tor, which is known to have high latency overhead for small webpages [2].

4.6 Proximity-based Authentication

In Android Lollipop, a user can register a smartwatch as "trusted", which means that Android will allow the user to bypass the lock screen if the device is nearby, where "nearby" is defined as within range for a Bluetooth Low Energy connection. To test this authentication protocol we used a Galaxy Gear Live smartwatch running Android Wear.

We used our idle test to establish a baseline with both WiFi and Bluetooth on, but no device connected.[6] We allowed both the baseline and smartwatch experiment to drain at least 90% of the battery, and then computed the average battery drain per hour. The device drained 3.71% per hour in the baseline and 3.79% per hour with the smartwatch connected, an energy overhead of 2%. Therefore, we conclude that the trusted device implementation in Android is energy efficient.

[6]This allows the operating system to put the Bluetooth radio to sleep, but would also represent a realistic comparison to the energy cost of "check-in" packets keeping a radio awake.

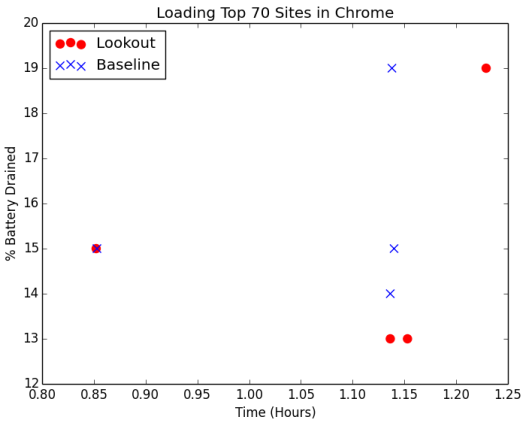

Figure 11: Energy usage and time of Lookout, opening pages in Chrome

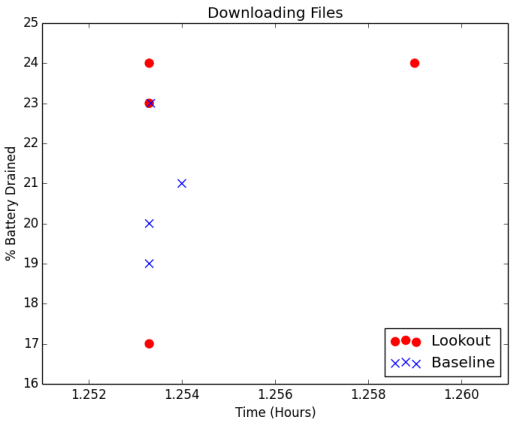

Figure 12: Energy usage and time of Lookout, downloading files

4.7 Antivirus Suites

Because Android does not allow an application access to arbitrary network traffic, running our web connections test suite for antivirus suites would not be appropriate. Therefore, we tested antivirus suites by loading pages in Chrome for Android, which allows antivirus applications to inspect traffic. We opened the same set of sites as before in Chrome to allow the antivirus suite to be active.

We first tested Lookout Mobile Security using this test suite, and as seen in Figure 11, the energy and time usage of having Lookout running in the background is within the distribution of tests for baseline. Because this test tended to be noisy due to more components running (loading up a separate browser application introduced unknown variables into our experiment), we conclude that the energy usage of Lookout in web browsing is negligible.

In addition, we also tested a set of downloads so that the antivirus suite could scan the download for safety. We obtained a dataset of downloads[7], and had our application issue download requests to the operating system. We left Lookout running in the background during this test so that it could scan the downloads. As seen in Figure 12, downloading files with Lookout running gave negligible differences in time, but a 6% increase in battery use.

[7]We used `https://github.com/ytisf/theZoo`, which is a dataset of malware designed for promoting malware research.

5 Conclusion

5.1 Discussion

One broad conclusion from our experimental results is that security protocols cause increased energy use insofar as they require the wireless radio and device screen to remain in high-powered states for longer periods of time. The latency overhead of security protocols is in an unhappy valley; it is long enough to be detrimental (especially since the wireless radio must remain powered while awaiting packets), but short enough that mitigative measures such as powering down the device screen while waiting for network activity are unfortunately unlikely to be practical.

5.1.1 Connection Reuse

With TLS, we observed that the energy overhead of a connection came from the TLS handshake. Therefore, we suggest that connection reuse and TLS session resumption should be highly effective means of reducing the energy overhead of TLS, as this will reduce the number of TLS handshakes required. To this end, we suggest that devices could proxy traffic through a single long-running TLS session. Such solutions are already deployed to minimize device bandwidth usage [31]; extending such systems to TLS proxying would provide users with security at low energy cost. While this solution would not provide end-to-end encryption as the proxy provider could read users' data, it would protect users from wireless eavesdroppers.

5.1.2 Limitations

Our proposed methodology of accounting for the energy costs of security protocols, as discussed above, is accurate but not precise. We are unable to measure the dynamic range in the energy use for a single connection; instead, we focus on determining aggregate energy use over a large number of connections. Therefore, following our current methodology cannot determine whether security protocols suffer from the "straggler problem" — that is, high tail energy use. If the straggler problem affects a significant number of connections, further work would be required to determine the cause of straggler connections and determine possible remedies.

5.2 Future Work

5.2.1 User Decisions

It remains unclear the extent to which users use their device's remaining battery life to make usage decisions. If users restrict device or choose to not use security protocols use when the battery is low, then measures that seek to minimize the energy cost of security protocols may convince users to use security measures at all times. Alternately, if users make decisions about whether to use security protocols without considering energy use, then while it would nevertheless be beneficial to maximize device battery life measures that specifically focus on mitigating the energy cost of security should be deprioritized. A user study is necessary to determine the appropriate course of action and furthermore could represent real-world scenarios better than our emulated tasks.

5.2.2 Precision

Because our instrumentation had limited precision, we were unable to measure energy use at a per-connection granularity. As a result, our study does not provide insight into the effects of system-level activity or network state except over timespans in the order of hours as these effects are averaged out in the aggregate. Future work using more precise instrumentation could study energy effects at a per-connection level to measure the variance in our readings.

5.2.3 Other Protocols

We were only able to perform a measurement of a limited set of security protocols in this work. Further work is necessary to apply our methodology to a broader set of security measures.

5.2.4 Future Protocol Design

Finally, we suggest that in the future security measures for mobile devices should be designed with their energy costs in mind. It is clear that the energy overhead can be significant, and energy efficient security implementations will benefit users.

References

[1] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, IETF, August 2008. `http://tools.ietf.org/html/rfc5246`.

[2] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *In Proceedings Of The 13th Usenix Security Symposium*, 2004.

[3] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Commun. ACM*, June 2004.

[4] D. Culler, D. Estrin, and M. Srivastava. Guest editors' introduction: Overview of sensor networks. *Computer*, Aug 2004.

[5] Chris Karlof, Naveen Sastry, and David Wagner. Tinysec: A link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, 2004.

[6] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 1999.

[7] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, 2004.

[8] A.S. Wander, N. Gura, H. Eberle, V. Gupta, and S.C. Shantz. Energy analysis of public-key cryptography for wireless sensor networks. In *Pervasive Computing and Communications. Third IEEE International Conference on*, March 2005.

[9] Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica. Quanto: Tracking energy in networked embedded systems. In *8th USENIX Symposium on Operating Systems Design and Implementation*, Dec 2008.

[10] A. Karagiannis, D. Vouyioukas, and P. Constantinou. Energy consumption measurement and analysis in wireless sensor networks for biomedical applications. In *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assistive Environments*, 2011.

[11] Frank Stajano and Ross J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proceedings of the 7th International Workshop on Security Protocols*, 2000.

[12] T. Martin, M. Hsiao, Dong Sam Ha, and J. Krishnaswami. Denial-of-service attacks on battery-powered mobile computers. In *Pervasive Computing and Communications. Proceedings of the Second IEEE Annual Conference on*, March 2004.

[13] M. Brownfield, Yatharth Gupta, and N. Davis. Wireless sensor network denial of sleep attack. In *Information Assurance Workshop. Proceedings from the Sixth Annual IEEE SMC*, June 2005.

[14] R. Racic, D. Ma, and Hao Chen. Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery. In *Securecomm and Workshops*, Aug 2006.

[15] Radhika Mittal, Aman Kansal, and Ranveer Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, 2012.

[16] Abhilash Jindal, Abhinav Pathak, Y. Charlie Hu, and Samuel Midkiff. Hypnos: Understanding and treating sleep conflicts in smartphones. In *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013.

[17] Panagiotis Vekris, Ranjit Jhala, Sorin Lerner, and Yuvraj Agarwal. Towards verifying android apps for the absence of no-sleep energy bugs. In *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems*, 2012.

[18] Ning Ding, Daniel Wagner, Xiaomeng Chen, Abhinav Pathak, Y. Charlie Hu, and Andrew Rice. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, 2013.

[19] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *In Proceedings of the 2010 USENIX conference on USENIX annual technical conference*.

[20] Neal Leavitt. Malicious code moves to mobile devices. *Computer*, December 2000.

[21] Simon N. Foley and Robert Dumigan. Are handheld viruses a significant threat? *Commun. ACM*, January 2001.

[22] D. Dagon, T. Martin, and T. Starner. Mobile phones as computing devices: the viruses are coming! *Pervasive Computing, IEEE*, Oct 2004.

[23] N. Leavitt. Mobile phones: the next frontier for hackers? *Computer*, April 2005.

[24] G. Lawton. Is it finally time to worry about mobile malware? *Computer*, May 2008.

[25] Hahnsang Kim, Joshua Smith, and Kang G. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, 2008.

[26] Johannes Hoffmann, Stephan Neumann, and Thorsten Holz. Mobile malware detection based on energy fingerprints - A dead end? In *Research in Attacks, Intrusions, and Defenses (RAID) - 16th International Symposium Proceedings*, Oct 2013.

[27] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. Analyzing the energy consumption of security protocols. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, 2003.

[28] Android developer sdk. `https://developer.android.com/sdk/index.html`.

[29] Alexa internet. `http://www.alexa.com`.

[30] Wireshark. https://www.wireshark.org.

[31] Victor Agababov, Michael Buettner, Victor Chudnovsky, Mark Cogan, Ben Greenstein, Shane McDaniel, Michael Piatek, Colin Scott, Matt Welsh, and Bolian Yin. Flywheel: Google's data compression proxy for the mobile web. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*.