2021

Pollen: A Secure, Decentralized Wireless Communication Platform

Max Perrello
Colby College

Pollen: A Secure, Decentralized Wireless Communication Platform

Max Perrello, *Student*, Ying Li, *Advisor*

Abstract—In this article, we detail an encrypted, internet-independent messaging platform, of which we built a prototype. This thesis also describes the challenges faced in the process, as well as potential future enhancements.

1 Introduction

On any given day, if one refers to netblocks.org, they can see a number of countries in which the government has decided it doesn't want its users accessing the free and open web. If you're an oppressive government, and you've found yourself reading this because you decided to enter a digital lockdown, you're in good company! Whether you're currently conducting a military coup, attempting to brainwash your citizens, or just doing some run-of-the-mill election rigging, you and the many, many other governments pulling this crap are in for a treat–I've built a platform to help your citizens circumvent your tyranny! In fact, to answer **@ajiswriting**'s question: yes—my thesis will defend me, and hopefully many others [1]!

2 Background and Motivations

Today's digital messaging, whether it runs on a peer-to-peer network or a centralized server, is still reliant on sprawling, interconnected networks. A broken link in these networks could cut users off from the rest of the world, and even make localized communication more difficult (think—when WhatsApp goes down, does it continue working locally, or is it down for everyone?). Finally, many communication platforms (e.g. Facebook Messenger) also relegate privacy to the back burner, or pretend the concept doesn't exist altogether. This is where my thesis comes in: Pollen is a cross-platform application that allows users to communicate securely, without reliance on the Internet or any alternative mesh network. Instead, messages are exchanged asynchronously, through a series of independent nodes. Messages propagate across the independent nodes in a process akin to that of the way bees pollinate flowers: any time a user interacts with a node, the user also receives unreadable (encrypted) copies of messages sent by other users, and then distributes them to a certain number of other nodes they interact with. Such a system is capable of functioning with any tiny number of users, and scaling to accommodate much more. If you're getting to this point with zero understanding of my project, that's okay—it will get both better and worse! Expect that this is not a conventional thesis: I will do my best to make things reasonably-comprehensible, but only when doing so is not at the expense of a real explanation.

• *Max Perrello and Ying Li are with Colby College.*

Manuscript received May 22, 2021.

3 Implementation

There are two major applications that run the Pollen platform: the Node and the Client. Each of these applications is broken down into a handful of major components, all of which are well-commented throughout the code and documentation. Here, we will stick to a reasonably-high level explanation, but the documentation should provide a fuller picture if so desired: Nodes, at least in their initial conception, are intended to run on existing wireless networks, like those found at coffee shops. Under normal circumstances, these networks provide public access to the Internet, but in an Internet shutdown situation, they still function as a local network, and can facilitate interaction between users who connect specifically to that wireless access point. Nodes are entirely independent and self-sufficient, but are also interoperable with other independent Nodes. Each one functions as a "dead drop," meaning they facilitate an asynchronous exchange of messages (see Fig. 1). In the simplest exchange, a sender (whom we'll call Client A) "drops off" their message at a Node, and Client B (the intended recipient) "picks up" the message from the same Node at a later time. In this example, the Node can be thought of as a simple mailbox, in which users leave messages for one another. However, in a more complex example, a Client has two responsibilities: sending and relaying messages. In this example, a sender (we'll call them Client A) drops off the

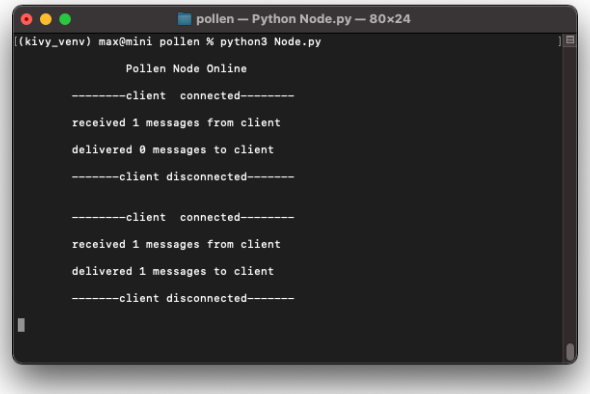

Fig. 1. An actively-running node, with message details.

initial message, intended for Client B. Another user (whom we'll call Client C), who happens to interact with the Node, picks up a copy of the message. Client C then drops the message off at a predetermined (as it stands, 10) number of other Nodes, meaning that Client B (the intended recipient of that message) can pick it up across a reasonably-wide geographical area. This exchange is analogous to the way bees pollinate flowers, hence the name of the app. It's important to note that none of these message exchanges happen manually—users send and receive messages the same way they would on WhatsApp, Signal, or iMessage (as seen in Fig. 2), it just so happens that nothing is transmitted unless one interacts with a node. This is also true of users who help

Fig. 2. Client home screen, with a number of active conversations.

propagate other users' messages on the network—when they interact with a node, they simply exchange their own messages, and the app handles the pickup and subsequent redistribution of messages entirely in the background. Further, these messages are fully encrypted in transit, using PGP [3], meaning that a malicious Client or Node on the network is theoretically powerless to read the contents of or otherwise alter a given message—the worst they can do is not deliver it. In practical terms, this means that the only person who can "read" a message is the person whom that message was "addressed" to, and because messages are also signed by recipients, modification/replacement of that message is impossible without detection.

4 Challenges

There were a number of challenges I faced in building this application. For the sake of brevity (as well as saving face—if you want to see the errors caused in part by my stupidity, see my initial presentation), I'll only detail the two most prominent technical challenges I faced: First and foremost, release compilation did not go well. This may be obvious to you when I tell you I was trying to build a fairly complex, fully cross-platform application in Python [4], but at this point, that's neither here nor there. I'll give a lot of credit to the Kivy community, who spent reasonably-substantial amounts of time helping me fix my build scripts. However, I do want to say that, in my view, aside from simple projects, it probably doesn't make sense to build any major mobile applications in Python, or at least not on the assumption that you'll be using Kivy to run the frontend and backend locally on each device. Needless to say, I spent a great deal of time trying to get my releases compiled, even after I thought I'd completely finished my app. Who knows, maybe that's par for the course—this is the first major application I've built myself.

The other major challenge I faced, and also haven't really solved, is data persistence. I'm ashamed to admit it, but as it stands, Pollen dumps a serialized instance of the client object to local storage, every time it's modified. Effectively, instead of efficiently storing messages and other app data in an organized table, I'm just saving the entire app, every time anything is modified. The story there is one of inexperience and lack of time—I'm aware that by storing unencrypted data locally (at least for Clients, the Nodes have no such access), I'm opening users up to vulnerabilities in which other, malicious applications can access that data on their device. Thus, while the data certainly needs to be stored properly in a database, I also want to make sure that that database is unreadable to all other local applications. For the sake of a proof-of-concept (especially when I do actually need to do a ground-up rebuild of everything else in order to build decent releases), using Pickle to dump the client data periodically seems relatively reasonable to me. So sue me. A more complete log of the challenges I've faced can be found in the issues on the Pollen GitHub Repository.

5 Future Enhancements

Finally, there are a number of avenues for potential enhancements, once the rebuild is complete (or someone somehow convinces me I should stick with Python). One of these expansions is the addition of Message Receipt Indicators. I anticipate some issues with nodes being overburdened by duplicate messages, since there is currently no way of communicating across the network that messages have been successfully delivered. This creates high potential for "orphaned" messages that stick around unreceived, until they are eventually deleted by the Node (30 days later).

This also worsens the platform's resistance to potential DOS (Denial of Service) attacks, in which malicious users could flood the network with useless messages, each of which is unable to be deleted for a given period of time. This leads into another potential enhancement: Trust Scoring/Certification—"hive" (groups of nodes) operators could sign off/vouch for nodes, using a process similar to my implementation of a decentralized vaccination passport [5]. At a high level, by deeming certain keys more trustworthy, and allowing them to sign off on other public keys, one can enable a sort-of vouching process for Nodes, and potentially for users as well. Taking advantage of Message Receipt Indicators is also a plausible option here: Clients could automatically sign off on a Node upon successful message exchange, enabling other Clients to build a trust score for Nodes. This creates the potential for trust score thresholds, which could be automatically set for less-technical users, in order to ensure that their messages are delivered. There are a lot of ways this could be expanded, each of which opens up other possible problems and methods for abuse. Another enhancement that I'd like to provide is an Alternative Syncing API, which would allow "hive" operators (administrators running groups of Nodes) to implement alternative message propagation/synchronization methods, without modifying the source code on a case-by-case basis. A few ideas for alternative syncing include: a LoRa intranet, BLE mesh network, direct to internet, etc. Networks like these might also take advantage of another undeveloped improvement: Public Bulletin Boards, on which a cache of unencrypted messages could be stored, syncing to every Client upon connection to a given Node. However, such a system offers particularly high potential for abuse, given that I'd like to minimize the rate at which disinformation and rumors can spread, like those caused in part by WhatsApp mass-forwarding and subsequent lynchings in India [2]. Finally, Substitute Installation Methods are a definite improvement I'd like to pursue. Given that there is scarce pre-planning in a situation that might require the usage of Pollen, it might be tough for users to actually install the app, given that conventional distribution methods mostly rely on Internet-based stores, like the App Store or Google Play Store. This also means that there needs to be a way for users to verify their installation, either with the help of other users or with Nodes, that could verify that an app version hasn't been tampered with, in order to avoid the distribution of malicious source code.

6 Conclusion

Overall, the purpose of Pollen is to provide a secure, decentralized alternative messaging framework—one that is not reliant on the Internet, for a number of reasons. My hope is that, despite my probably-poor implementation, by creating a fully-open source project like this, and by embedding my relatively-novel ideas, I will open avenues for those better than me to learn from my mistakes and create a system that will have a net positive impact on the world. Here's to hoping.

Appendix
Additional Screenshots

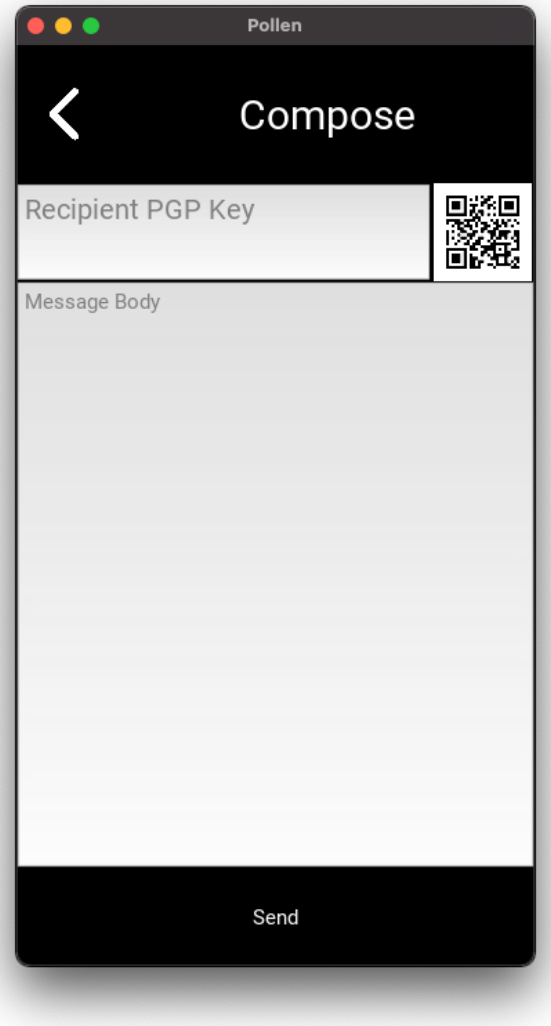

Fig. 3. Client compose screen.

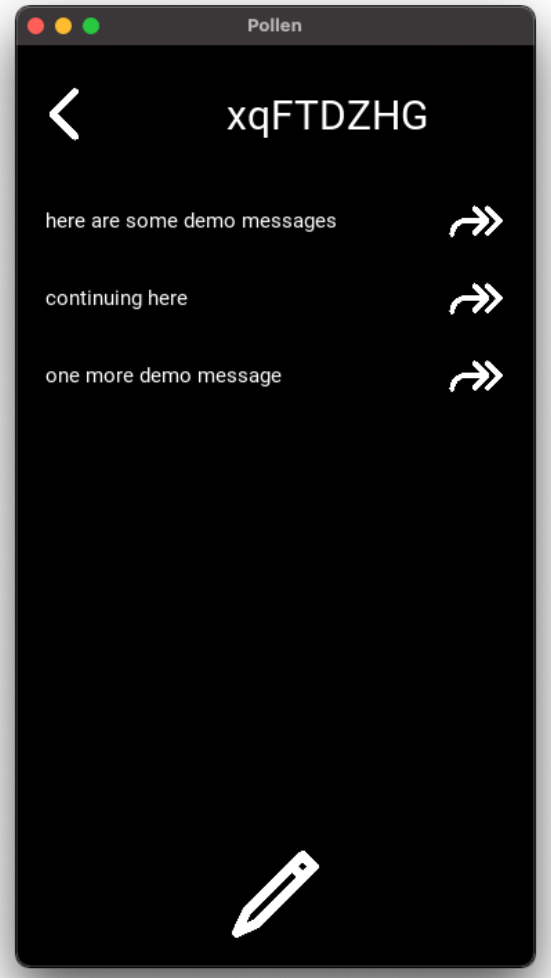

Fig. 4. Client conversation screen, with a few outgoing messages.

Acknowledgments

The author would like to thank those who think the solution to valid criticism is silencing those who speak up.

References

[1] A. Addae [@ajiswriting], but would your thesis defend YOU?, *Twitter*, April 5, 2021. [Online]. Available: https://twitter.com/ajiswriting/status/1379163432310673409, Accessed on: April 6, 2021.

[2] *"How WhatsApp helped turn an Indian village into a lynch mob,"* Jul. 19, 2018. Accessed on: Apr. 20, 2021. [Online]. Available: https://www.bbc.com/news/world-asia-india-44856910

[3] M. Greene, GitHub. 2014. PGPy, ver. 0.5.3. [Online]. Available: https://github.com/SecurityInnovation/PGPy

[4] Kivy Team and other contributors, GitHub. 2011. Kivy, ver. 2.1.0.dev0. [Online]. Available: https://kivy.org

[5] S. Arora, G. Cahill, M. Kielstra, and M. Perrello, Devpost. 2021. VaxPort, ver. 0.0.1. [Online]. Available: https://devpost.com/software/vaxport