# PANCAKE: A Central Management Authority for Coordinating a Personal Area Network and Controlling External Access

Tamara Denning
Computer Science & Engineering
University of Washington

**Abstract**
In this paper I present the PANCAKE, a device for managing and protecting a low-power personal area network. I then describe an emulation of the system written on the Seattle platform. The paper analyzes the security properties of the system and then presents some data on the implementation's performance. Broadly speaking, the PANCAKE provides attractive security benefits, but this implementation suffers from a communication bottleneck.

## 1. Introduction

Current research into mobile hardware, personal sensors, and other ubiquitous technologies suggests the possibility that, in the future, individuals will be instrumented with a number of personal devices. These diverse devices could benefit from having a central management device that coordinates inter-device communication and oversees access to devices from third-party clients. In this paper I propose such a device, which I call the Personal Area Network Coordinator and Access KontrollEr (PANCAKE), and explore some of its basic properties.

The PANCAKE's utility can be summarized into two broad categories: the PANCAKE, being battery-powered and rechargeable, has more available power than some personal devices, and can therefore be used to handle more power-intensive tasks; and it simplifies users' management tasks so that they have only a single entry point for configuration. To illustrate the PANCAKE's usage model and how it might benefit a Personal Area Network (PAN), I will give an example.

**A usage scenario.** One personal device that might be in a user's PAN is a sensor that periodically polls air quality. This sensor allows the user to collect information about the air in the environments that he visits throughout his day. The user may wish to download this data wirelessly to his home server at the end of the day, where it is processed and summarized in a graph. Additionally, he may wish to grant read-access to the city's sparsely located environmental nodes (they were deployed as part of an initiative to collect long term statistics about environmental conditions in the city, and are located in areas that receive a large amount of pedestrian and vehicular traffic). The user does not want any other parties to read this data, since he is concerned that this might violate his privacy by revealing approximately where he has been.

The air sensor is powered by a battery, but the user wants to charge the battery as infrequently as possible, so we would like to limit the power-consuming operations on the sensor and when possible transfer tasks to the PANCAKE. (Some devices in the PAN might be passively powered and therefore incapable of performing power-intensive tasks.) We can limit the amount of power consumed by the sensor by causing its transceiver to go to sleep when not transmitting; in this way, the sensor will not waste power listening to local wireless traffic. It is still desirable, however, to be able to access data on the air sensor when

necessary. One way to approach this problem is to equip the sensor with a passively-powered chip that will wake up the transceiver only when it receives correct authentication (the feasibility of which is demonstrated by Chae *et al.*) [2]. Let us suppose that the city advertises public keys for its official environmental nodes so that citizens can verify that a communication is legitimate; it is currently unrealistic to suppose that a passively-powered chip could authenticate a reader in this manner. Instead, the PANCAKE can maintain a store of public keys and authenticate a third party that is trying to access a user's personal device. Once the client is authenticated, the PANCAKE—which has a long term pairing with devices in the PAN—can wake the sensor's transceiver up using symmetric cryptography. At this point, the PANCAKE serves as a proxy between the client and the personal device, logging all communications between the two.

The PANCAKE can also support usage of the air sensor in other ways. For example, the air sensor could notify the PANCAKE if it senses particularly poor air quality, and the PANCAKE could issue an audio, tactile, or textual notification to the user. Alternatively, if the air sensor runs out of local storage, it could use the PANCAKE for additional storage.

**Other uses.** Since the PANCAKE functions as the gateway to the PAN, it has the potential to serve as an inter-protocol translator. There is no need for a client and a personal device to communicate using the same protocol, as long as the PANCAKE has knowledge of a mapping between the two protocols.

**2. System Emulation**

I implemented a basic emulation of the PANCAKE system in approximately 650 lines of code across four files [1]. Three of the files emulate the behavior of the PANCAKE, a personal device in the PAN, and a third-party client, while the fourth file contains functions that are common to the three modules. The unique properties of specific personal devices or clients are loaded into an instance of their module using configuration files. Communications in the system are implemented via TCP sockets. While the communications were originally designed to transmit between hosts at different IP addresses, where each host simulates an entity in the system emulation, I found that it was smoother to develop the emulations as separate processes on the same host.[1] Entities in the system are assigned distinct port numbers and are configured to listen for TCP communications on that port.

The PANCAKE keeps information about each of its paired devices, including the device's name or identifier, the type of the device, the private key shared with the device, and the port on which to contact the device. For example:

```
device0.port=12347
device0.type=air
device0.key=255,144,0,133,252,147,162,70,31,168,176,32...
```

The PANCAKE also keeps a list of clients about which it has knowledge, along with relevant details such as the port on which to access the client and the client's public key.

The PANCAKE is designed to process both clients requests that are addressed to one particular device in the PAN and requests that are broadcast to a more general audience. In

---

1. For example, It is much faster and easier to watch the output of a process locally in real-time than to repeatedly query a remote host to monitor the progress of a node.

the latter case, the client request is presumed to be directed towards a certain type of device (e.g., an air quality sensor or an RFID ID card), and the PANCAKE delivers the request to all devices in the PAN of that type.

Since this system design aims to provide increased security guarantees while minimizing the computational and power demands on devices in the PAN, the PANCAKE takes care of verifying the authenticity of client requests and encrypting responses with the client's public key. Messages destined for personal devices are encrypted with the shared private key and sent on to the device.

## 3. Security Properties & Analysis

The system emulation has a number of security precautions in place to help protect the confidentiality and integrity of the system, as well as to help ensure the authenticity of message origins. The implementation has not been subjected to penetration testing; instead, I describe the security precautions below and discuss their effects on the system's properties.

### 3.1. Confidentiality

Confidentiality in the system is protected by a combination of public-key and private-key cryptography. As mentioned above, the PANCAKE and the devices in the PAN have a long term pairing and share a symmetric key (each PANCAKE-device pair has a unique key). In order to prevent eavesdropping, all communications between the PANCAKE and a device are encrypted using an AES implementation in the Seattle library.[2] Similarly, the PANCAKE protects the privacy of its messages to a client by encrypting the message with the client's public key, using an RSA implementation in the Seattle library. Untargeted queries in the system emulation (e.g., a reader queries the surrounding area for any devices willing and able to respond) send their queries in plaintext.

The emulation implementation provides additional confidentiality by incorporating a timestamp into every message's plaintext, thereby perturbing the ciphertext resulting from otherwise identical plaintexts.[3]

### 3.2. Authenticity

Third-party clients in the system authenticate themselves to the PANCAKE by signing their messages. More specifically, the client produces a hash of the plaintext message using a SHA implementation in the Seattle library, signs this hash using its private key, and transmits this signed hash along with the message. This method gives the PANCAKE reasonable confidence that the message in question originated from the client. The PANCAKE does not sign its messages to clients, since in this model we are concerned about the privacy and security of the PANCAKE and the devices in the PAN; the implementation could easily be changed so that the PANCAKE also authenticates itself to clients by signing its messages. The PANCAKE and the PAN devices do not sign their messages to each other, since they only use symmetric cryptography on their communications. Instead, a measure of authenticity is provided by encrypting the message hash, as mentioned in the next subsection. This

---

2. Thanks, Justin!
3. This is particularly important because, for the sake of simplicity, I use the same initialization vector for every encryption operation. This is a potential weakness, as will be discussed later.

method does not distinguish between the identities of the PANCAKE and the communicating device, but since the PANCAKE and a personal device trust each other in this system model, there is no need to support non-repudiation.

### 3.3. Integrity

The implementation incorporates message integrity codes (MICs) to provide some assurance that a message has not been tampered with or damaged in transit. All entities hash the message's plaintext prior to transit using a SHA implementation in the Seattle library. As previously mentioned, a client with a broadcast transmission then signs this hash with its private key. Communications between the PANCAKE and its devices encrypt the hash with their private key and send the encrypted hash along with the encrypted message. These measures provide a high degree of assurance that the sending entity did send the plaintext as received.

### 3.4. Replay Attacks

The emulation attempts to prevent replay attacks by incorporating a timestamp into the plaintext of every message. Each entity keeps track of the last time that each other entity has sent them a message, and only accepts messages that have timestamps that occur later. In this way, no entity will act on an encrypted message that has been replayed by an adversary.

### 3.5. Potential Weaknesses

The current implementation provides no protection against Denial-of-Service (DoS) attacks. An increased number of incoming requests will cause the PANCAKE to respond more slowly to or drop legitimate requests. There are some changes that could be made to the implementation to improve its security properties. These changes include a scheme for changing the initialization vector's value over time and negotiating temporary session keys to use instead of the long term keys.[4] Both of these changes would decrease the probability that an adversary will be able to recover the keys used by decreasing the predictability of messages. In the current implementation, an adversary with knowledge of the commands that can be issued to a device (e.g., "read setttings") and the translation between client-PANCAKE and PANCAKE-device protocols could potentially recover their private key. If the adversary has some legitimate access to the device, he can perform a plaintext attack by issuing commands to the device. In combination with trying various timestamp values near the time when the message was sent, the adversary could successfully recover the key.

## 4. Performance

While performance optimization was not a goal in the implementation of the PANCAKE emulation, performance metrics provide an idea of how viable the system would be for actual use. Particularly, it is of interest how many simultaneous devices and client requests the PANCAKE can handle. There are a number of factors that affect the workload on the PANCAKE, including:

- How many clients communicate with the system
- How often clients communicate with the system
- Whether client messages are spaced out or are clustered together
- Whether client messages are targeted for one device or for a number of devices

---

4. For example, the Diffie-Hellman protocol could be used to negotiate session keys.

In order to get a rough idea of system performance, I ran tests that measured system latency in response to manipulating the above factors. Figure 1 and Figure 2 show results from the tests.

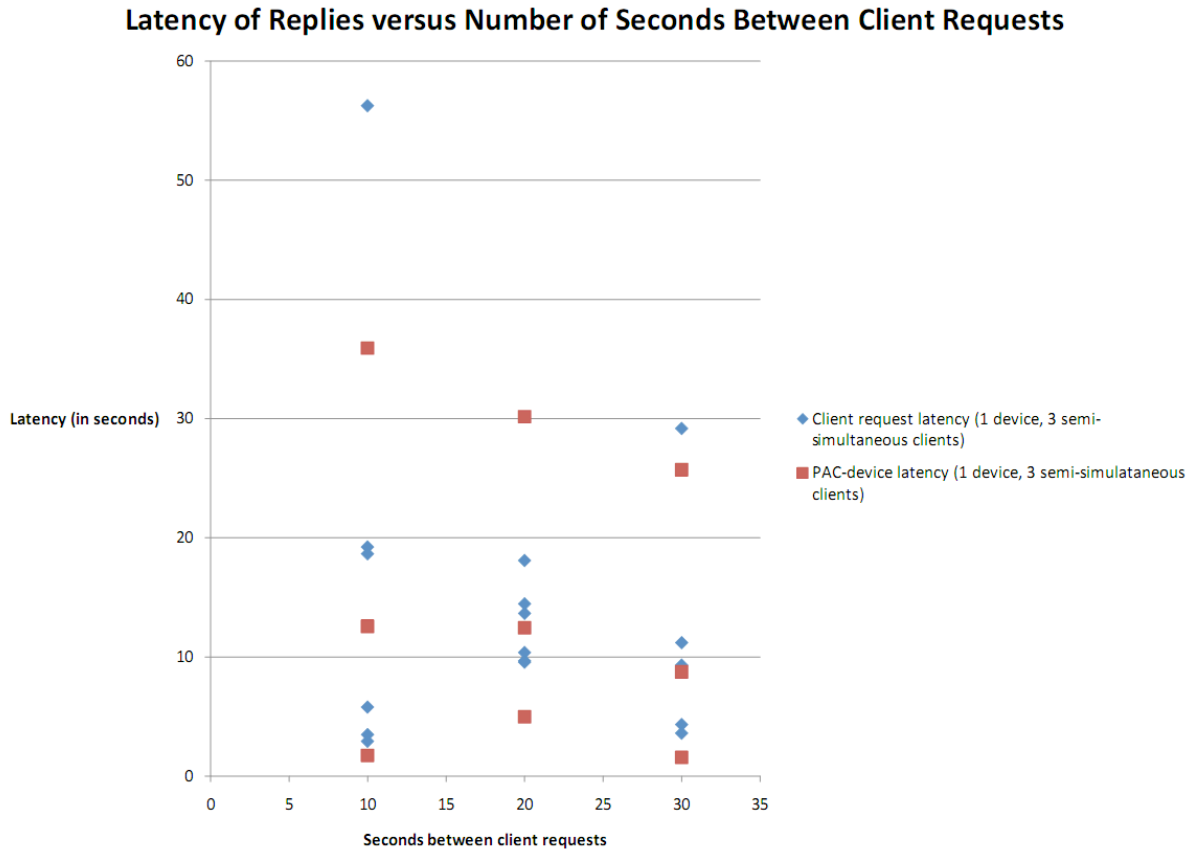


Figure 1. Above are plotted the average latencies between sending a message and receiving a response, averaged over a 2-minute period, for a system in which there are three clients, one PANCAKE, and one device registered with the PANCAKE. The x-axis indicates the rate at which the clients in the system send out messages. The different point graphics correspond to the latencies between the PANCAKE and its devices and the latency between the time when a client sends out a message and receives a response from the PANCAKE.

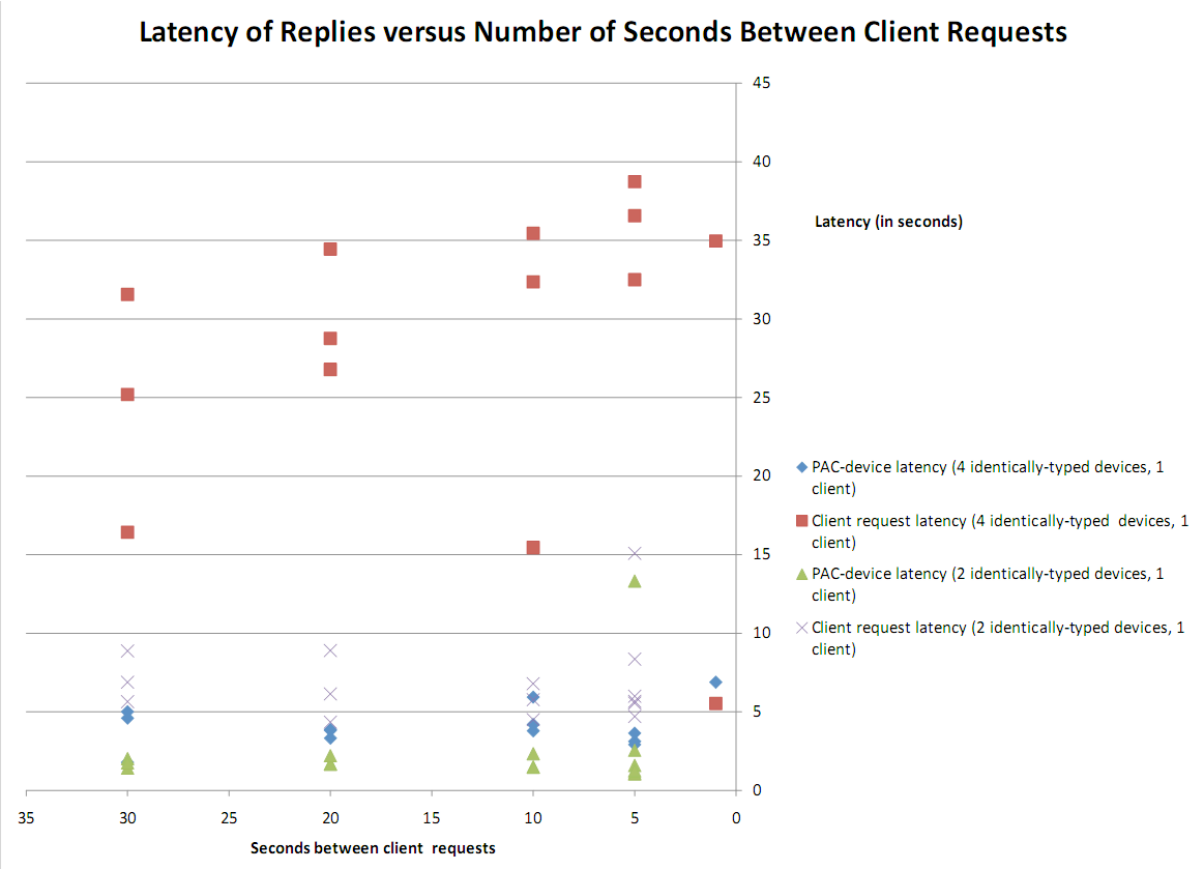


Figure 2. Above are plotted the average latencies between sending a message and receiving a response, averaged over a 2-minute period, for a system in which there is one client and one PANCAKE. The x-axis indicates the rate at which the client sends out messages. The different point graphics correspond to the PANCAKE-device and client-PANCAKE-device-PANCAKE-client latencies. The graphics further distinguish between two system configurations: one in which there are 4 devices of the same type (and therefore all need to respond to the same client requests) and one in which there are 2 devices of the same type.

In Figure 1, points for 5 second- and 1 second- request intervals are omitted from the plot because the system started failing to transmit messages, presumably due to overload. The remaining results in Figure 1 are erratic, but unfavorable. Although I do not know why the latencies vary so widely in the plot, I suspect that it might be due to the fact that the synchronization of the clients varied from trial run to trial run; although clients were started at the same time, they did not always stay synchronized, and therefore might have contributed unevenly to system congestion.

In Figure 2, the results are easier to interpret. There are several trends in the graph that support expected system behavior: client request latencies are larger for a given system configuration than PANCAKE-device latencies; a larger number of devices with identical types, and who therefore respond to the same client messages, increase system latencies; and—although less clearly—shorter client request intervals correspond to larger system latencies. Judging from Figure 2, an emulation with 4 identically-typed devices has an unacceptable latency time, but an emulation with 2 identically-typed devices has acceptable performance for non-critical applications.

## 5. Discussion

As previously mentioned, one of the advantages of the PANCAKE design is the fact that it is a central management point, which could simplify user management tasks and provide inter-protocol translation services. Additionally, the power resources available to the PANCAKE—as opposed to passively-powered or power-conscious devices in the PAN—allow the PANCAKE to enhance the security properties of the PAN.

One of the obvious disadvantages of the PANCAKE system design is that, by using a single device as a coordinating agent, the system is vulnerable to a single point of failure; if the PANCAKE is intentionally or accidentally disabled, there is no way to access devices in the PAN until the PANCAKE is brought online. Additionally, while the PANCAKE is able to remain operational during DoS attacks by arbitrarily dropping requests, DoS attacks can still cause some interruption of service to legitimate clients. Similarly, aside from DoS attacks or accidental failures, system performance is limited by the rate at which the PANCAKE can receive, process, and send messages.

## 6. Conclusion

In this paper I present the basic system design of the PANCAKE, a device meant to coordinate a personal area network of devices by centralizing management tasks and providing increased security. I describe an emulation of the system implemented on the Seattle platform, then discuss the implementation's security properties. Section 4 presents some latency data points using different numbers of clients, devices, and varying the client's request frequency. The emulation of the system performs acceptably when the PANCAKE is not attempting to receive or transmit many messages simultaneously; however, performance degrades quickly as the message load on the PANCAKE grows. While the central management point of the PANCAKE offers advantages—particularly security advantages—the system needs to achieve better performance before it can be considered competitive compared to more distributed solutions.

## References

[1] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: A Platform for Educational Cloud Computing. In *Proceedings of SIGCSE 2009*, pages 111-115, March 2009.

[2] H.-J. Chae, D. J. Yeager, J. R. Smith, and K. Fu. Maximalist Cryptography and Computation on the WISP UHF RFID Tag. In *Proceedings of Conference on RFID Security 2007*, July 2007.