Using Wake-on-LAN to Manage Home Devices

Thomas Smith
tcs1g20@soton.ac.uk
University of Southampton

***Abstract*—Idle devices result in a large amount of wasted energy within households every day. Through the Wake-on-LAN standard, this paper proposes a convenient way to power your devices on based upon environmental factors. This includes scheduling times, detecting nearby devices, and based upon environmental factors. Users interact with the application using a web API, which allows for easy remote access and control of home devices.**

I. Introduction

It is a common scenario for home devices to be left running all day despite only being in use for a fraction of the time they are awake [1]. The Wake-on-LAN standard offers the ability for many devices to be left in low-power modes and woken using a single central node; by watching for several different events in the local network and environment this project will be able to wake devices exclusively when they are required. By having this central node as a low-power device, this will result in a potentially large reduction in power usage.

Some example use-cases of this application are:

- Scheduling devices to be woken at recurring periods when a device is needed to be used. This allows the user to have a predictable timetable for when the device should be active.
- Waking a device when the presence of another device is detected locally. The presence of a local device can infer that the user is nearby and will want to use a different device.
- Similarly, waking a device based upon environmental factors can infer the presence of a user but does not require the user to have a mobile device that would need to be detected.

II. Goals

Table I shows the goals for this project that are organised using MoSCoW prioritisation to determine what objectives are most important to this project.

No.	Goal	Priority
1	Wake a Wi-FI device over a local network by sending a magic packet.	M
2	Allow devices to automatically be woken based upon external events.	M
3	Detect devices present in the local WiFi network.	M
4	Detect local Bluetooth devices that are in discoverable mode.	M
5	Allow users to interact with the application using a web HTTP API.	M
6	Use a local database to record any user data.	M
7	Use sensors to track environmental factors.	C

TABLE I
The goals of this project using MoSCoW prioritisation.

III. Research

A. Wake-on-LAN

Wake-on-LAN is a network standard that allows remote devices to be turned on by sending a *magic packet* to it. It was first introduced in 1998[1] by the Advanced Manageability Alliance (AMA); this alliance consisted of several large tech companies and was created with the purpose of introducing standards that would helped streamline computer management. Some examples of standards introduced by the AMA are: Desktop Management Interface[2], Alert Standard Format[3], and Common Information Model[4].

1) Magic Packet: A magic packet is a frame with a 102 byte payload that consists of:

- 6 bytes that are all *0xff*, and
- 16 copies of the target device's MAC address.

The target device's network interface card (NIC) will be listening for this packet whilst in a low-power mode; when it is broadcast over the network the target device's NIC will send a command to the power supply or motherboard to wake the system up.

2) Limitations: The Wake-on-LAN standard doesn't include any form of delivery confirmation so there is now way to know if the magic packet sent is received or acted upon.

B. Waking Non IP-Based Device

Wake-on-LAN uses an IP packet and as such cannot be sent to devices that don't use IP; these are commonly IoT devices and will use protocols like Zigbee to communicate. Han et al looked at using Zigbee to control power outlets in their paper *Remote-controllable and energy-saving room architecture based on ZigBee communication* [2]; devices can be toggled using an IR remote or will automatically be turned off when the power output from the socket falls below a certain threshold.

[1]https://web.archive.org/web/20121012155338/http://www-03.ibm.com/press/us/en/pressrelease/2705.wss (Accessed 09/05/2023)

[2]https://www.dmtf.org/standards/dmi (Accessed 09/05/2023)
[3]https://www.dmtf.org/standards/asf (Accessed 09/05/2023)
[4]https://www.dmtf.org/standards/cim (Accessed 09/05/2023)

C. Device Discovery

1) Over Wireless LAN: If we are aware of the device's IP address then it is trivial to discover if it is active by sending it a simple request, such as an ARP request, and wait for a response.

Zhou et al describe a system called *ZiFi* [3], which uses Zigbee radios to identify the existence of local WiFi networks; this was with the aim of reducing the power requirement needed for a device to discover local WiFi networks. Whilst the use of Zigbee is promising for device discovery, most devices will not come fitted with Zigbee radios.

2) Bluetooth: Bluetooth is a short-range, low-bandwidth, low-power wireless communication protocol that is commonly found in battery powered devices. Bluetooth devices organise themselves into *piconets*, which consist of one *master* and up to seven *slave* devices. Each device in the same piconet will have the same frequency hopping sequence that will determined by the master. The formation of a piconet has two steps:

1) **Inquiry** A master device will discover neighbouring slave devices, and
2) **Page** Connections are established between devices.

During the inquiry phase, the master will broadcast an inquiry packet and scan for replies. If a slave wants to be discovered then it will periodically scan for inquiry packets and send a response if they find one. In their paper *A formal analysis of bluetooth discovery* [4], Duflot et al give a review of the performance of device discovery in Bluetooth v1.2 and v1.1. A similar analysis of Bluetooth 4.0 is given by Cho et al in their paper *Performance analysis of device discovery of Bluetooth Low Energy (BLE) networks* [5].

Cross et al showed it was possible to discover devices that are not scanning for and replying to inquiry packets in their paper *Detecting non-discoverable Bluetooth devices* [6]. This was achieved using an 'enhanced brute force search attack'; this relied on already knowing the device's MAC address but would still take a considerable amount of time, where most Bluetooth devices would take around 18 hours.

IV. Design

This section will detail the design of each of the main elements of the application; this includes: how the application is interacted with, how data is stored, and the services being ran.

A. API

This application should provide a set of HTTP endpoints that allow users to perform various CRUD operations. This includes:

- creating and managing a set of devices that can be woken up using Wake-on-LAN,
- creating and managing recurring schedules for these devices to be woken up at,
- creating and managing a set of devices that can be watched by the application in order to trigger devices to be woken,
- create mappings between watched devices and devices that can be woken, and
- managing a set of background services that are used to track watched devices and environmental factors.

B. Services

This section will describe the background services that are used to detect various conditions that when satisfied will trigger a set of devices to wake up.

1) Schedules: This service will allow users to set recurring periods of time every week that a device should be woken up. Each schedule will detail:

- what device should be woken up,
- what day and time every week it should be woken up, and
- whether or not that schedule is active at this point in time.

2) Scanning the LAN: This service will periodically look for a *watcher* device on the local network by sending an ARP request to it and seeing if a response is received. An example use case for this would have a user setting their mobile device as a watcher so when they join their home network all of their devices will be woken up.

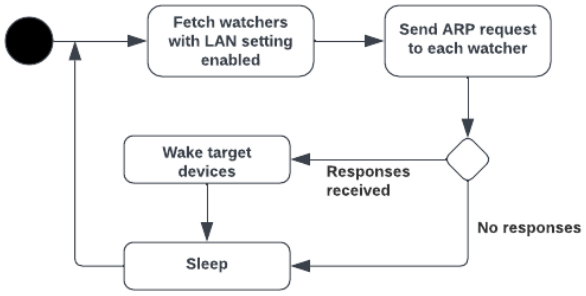

Fig. 1. A state diagram showing how devices are searched for on the LAN

Watcher devices will be checked for at regular intervals until they are discovered, and once they are found they will not be checked for again for a period of time specified by the user. This service will add extra congestion to the local network and relies on a device having a static IP address; most routers will use DHCP so users will have to set this manually.

3) Bluetooth: By allowing this application to search for Bluetooth devices, it increases the number of devices we can search for and removes the requirement from the previous section of being in the same LAN. On top of this, the watched device is referred to by its unique Bluetooth MAC address, which will never change, as opposed to a device's IP address, which will likely be allocated using DHCP and may change.

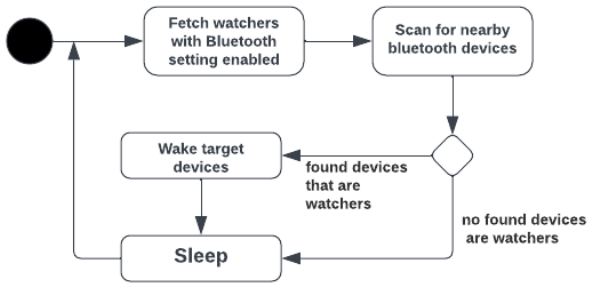

Fig. 2. A state diagram showing how devices are searched for using Bluetooth

4) Sniffing Traffic: This service will sniff for packets being sent to a watcher device and if the packet meets a given set of conditions then we will wake a target device. An example use case for this would be a user remotely connecting to a home server and then having the application wake up any auxiliary servers, such as a file server.

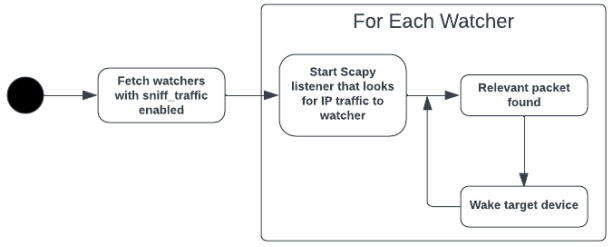

Fig. 3. A state diagram showing how devices are searched for using LAN packet sniffing

5) Environment Sensors: This application will have a set of services that use external sensors to consider various environmental factors; specifically: light, sound, and motion. The user will be able to set threshold values for each, such that when the sensor records a value that is above or below it a target device is woken up.

C. Arc

Figure 4 shows a sample architecture for the application; where the user is accessing the API remotely over the internet. A watcher device joins the network, is detected, and the application then wakes up a different device.

V. Implementation

This section will detail specific implementation details about the application created.

A. API

The API was written using the FastAPI[5] framework due to its simple syntax, built-in support for data validation and serialisation, and its support for OpenAPI documentation that allows developers to easily write documentation and have it hosted to allow users to explore the API. The use of OAuth2 with JSON Web Tokens meant the API endpoints are secure and only usable by registered users, which allows for the safe, public hosting of this API.

[5]https://fastapi.tiangolo.com/ (Accessed 09/05/2023)

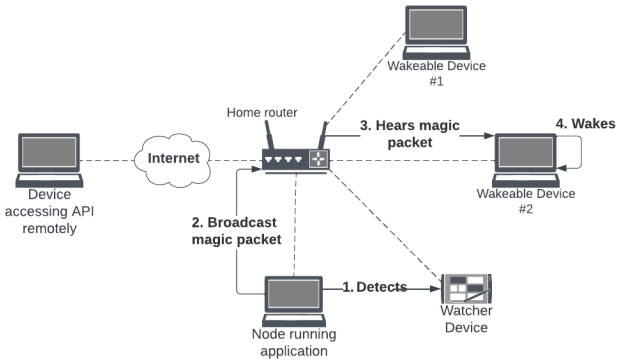

Fig. 4. A sample architecture for the application.

B. Data Storage

This application will create a local SQLite[6] database that will be interacted with using the object relational mapping library SQLAlchemy[7]. The following tables will be created:

- **wakeable_devices** These are the IP-based devices that can be woken over the network by sending a magic packet to them.
- **schedules** These relate a device to many recurring points in time at which the device should be woken up by the application. Each schedule can be toggled individually.
- **watcher_devices** These are devices that can be searched for locally using the methods described in Section IV-B. Each device will have a timeout that allows users to restrict how often a detection of this device triggers devices to wake up.
- **watcher_wakeable_mapping** This table shows the many-to-many associations between wakeable and watcher devices. Each mapping also stores which mediums should be watched; for example, should the application look for it using Bluetooth.
- **users** This will store the user information to help secure this API. This application will only consider a single admin user.
- **services** Stores information about the background services a user can run and whether they should be run on application startup. The services available are detailed in Section IV-B and each can be toggled on or off at any point.
- **environment_listeners** Stores information about each environment listener including what it is watching (noise,

[6]https://sqlite.org/index.html (Accessed 09/05/2023)
[7]https://www.sqlalchemy.org/ (Accessed 09/05/2023)

light, or temperature), what the threshold value is (to determine when to wake the device), whether this threshold is an upper/lower limit and a reference to the device being woken.

C. Other Details

1) Docker: For easy setup, a Dockerfile is provided to allow a user to start the application with a single command without having to manually install any of the dependencies.

2) Environmental Sensing: The Enviro[8] kit and the corresponding Python library[9] was used in conjunction with a Raspberry Pi 4 to detect various environmental factors.

D. Application Diagram

Figure 5 shows a breakdown of the application architecture. To help improve the maintainability, the applications is broken down into different layers that are abstract from each other, such that any of them could easily be replaced.

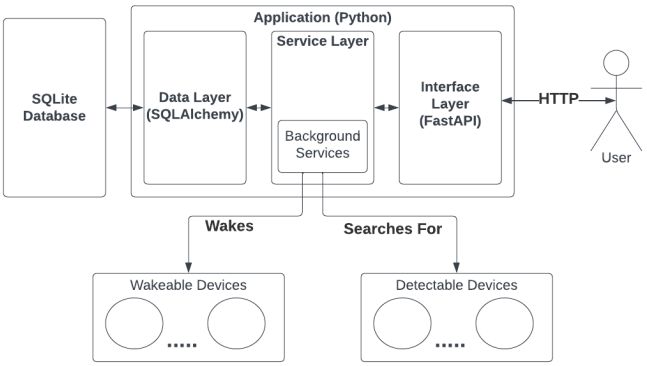

Fig. 5. A diagram showing the application architecture.

VI. Limitations

Some of the limitations found from this project include:

- devices cannot be remotely turned off through the application and instead rely on the user being able to turn it off themselves,
- Wake-on-LAN has no delivery confirmation so my application includes no method for users to verify whether a device was woken; instead it is up to the user to find out themselves, and
- The environment sensors used for this application are tightly coupled to a specific piece of hardware for a Raspberry Pi.

[8]https://shop.pimoroni.com/products/enviro?variant=31155658489939 (Accessed 10/05/2023)

[9]https://github.com/pimoroni/enviroplus-python (Accessed 10/05/2023)

VII. Conclusion

In conclusion, this project offers a convenient way for devices to be remotely and automatically woken from low-power modes, which helps contribute to the reduction of unnecessary power usage from idle devices.

An interesting next step would be to consider how this concept could be applied to non-IP devices; this would be specifically prevalent given the recent popularity of IoT devices that are typically *always on*.

References

[1] B. Urban, V. Shmakova, B. Lim, and K. Roth, "Energy consumption of consumer electronics in u.s. homes in 2013,"

[2] J. Han, H. Lee, and K.-R. Park, "Remote-controllable and energy-saving room architecture based on ZigBee communication," vol. 55, no. 1, pp. 264–268. Conference Name: IEEE Transactions on Consumer Electronics.

[3] R. Zhou, Y. Xiong, G. Xing, L. Sun, and J. Ma, "ZiFi: wireless LAN discovery via ZigBee interference signatures," in *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pp. 49–60, ACM.

[4] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker, "A formal analysis of bluetooth device discovery," vol. 8, no. 6, pp. 621–632.

[5] K. Cho, G. Park, W. Cho, J. Seo, and K. Han, "Performance analysis of device discovery of bluetooth low energy (BLE) networks," vol. 81, pp. 72–85.

[6] D. Cross, J. Hoeckle, M. Lavine, J. Rubin, and K. Snow, "Detecting non-discoverable bluetooth devices," in *Critical Infrastructure Protection* (E. Goetz and S. Shenoi, eds.), vol. 253, pp. 281–293, Springer US. Series Title: IFIP International Federation for Information Processing.