LOADHoC: Towards the Automatic Local Distribution of Computation Using Existing IoT Devices

Shaine Christmas[1](✉), Kevin Lee[1], and Jean-Guy Schneider[2]

[1] Deakin University, Geelong, VIC 3220, Australia
schristmas@deakin.edu.au
[2] Monash University, Clayton, VIC 3800, Australia

Abstract. Mobile devices often use offloading nodes to reduce the amount of power used locally, or because of the low computational power of the device. Mobile devices can take advantage of offloading to reduce the power usage and increase the battery life of the device. Cloud offloading architectures are largely out of the users control and have higher latency than running the computation on the local device. Cloud computing architectures use specialised nodes, which need to be designed for the service provider. As more Internet of Things (IoT) devices are created and deployed, the number of idle devices also increases. These devices use power to stay in an idle state, which constitutes an inefficient usage of computational resources; both as network resources and physical materials used to make the device. To address an alternative to cloud offloading architectures, this work proposes an architecture for taking advantage of idle computing power of IoT devices for offloading computations from devices locally to improve privacy, latency, and sustainability. Testing of the proposed architecture demonstrates that computations can be successfully offloaded, with acceptable latency, and minimal increase to the individual power use of the connected IoT devices.

Keywords: Internet of Things · IoT · Offloading · Distribution

1 Introduction

Mobile devices often require aid in completing computations, due to stringent power and computation needs [21]. Offloading uses existing computational infrastructure to enhance the capability of resource-constrained (mobile) devices. Depending on connection speeds and data privacy, using cloud-situated resources for offloading is not always desirable or possible. Local offloading methods can be used to address these concerns. This requires an additional device for the offloading of computation, which is not always required for smaller offloading applications. By taking advantage of locally deployed devices, idle computation can be used to effectively complete offloading requests within a timely manner, and with no data exposure to the wider Internet.

A. Zaslavsky et al. (Eds.): MobiQuitous 2023, LNICST 593, pp. 175–200, 2024.
https://doi.org/10.1007/978-3-031-63989-0_9

Within the wider area of IoT, many reduced functionality devices resort to offloading techniques and technologies to reduce local power consumption, or to circumvent local computation limitations. A common technique [31] of cloud offloading systems allows for devices to send computations to external services, and receive results given a short latency and run time. Whilst this works for applications where devices are connected to the wider internet, localised offloading techniques are still being explored for efficiencies in latency, reduction of communication complexity, and practical applications.

In addition to optimisation of offloading techniques for local applications, IoT devices are often produced with more computational power than strictly required for average operation [29]. In the context of Smart Home devices, many do not constantly use their full computational power, as their computation requirements vary largely. As the number of these devices increases, and become more complex [12], materials will become more scarce and difficult to manufacture. Use of local existing IoT devices would reduce the material cost of creating local offloading solutions.

Single device offloading systems can be a single point of failure for physical and digital vulnerabilities. Edge Computing [6] is the practice of keeping resources available to client devices as close to the client network as possible; often placing devices within the clients network for local access. Taking advantage of Edge Computing, a distributed architecture of devices to handle offloading on the edge would reduce latency, and increase privacy.

The aim of this work is to justify and propose an architecture for localised offloading of computations from reduced functionality devices to a network of low power devices. The contributions of this work are: (i) an architecture for dynamic offloading using pre-deployed IoT devices; (ii) a method for offloading with dynamic network sizes and computation sizes; (iii) an approach for negotiation of resources for multiple simultaneous clients; (iv) an evaluation of the architecture, focusing on scalability of devices, computation and offloading techniques.

The remainder of this paper is as follows: Sect. 2 looks at relevant past works. Section 3 presents the research design and methodology. Section 4 explores the areas of the proposed architecture to evaluate, and outlines the results of the experiment evaluation, to be further evaluated within Sect. 5, with Sect. 6 concluding this work.

2 Background

The Internet of Things (IoT) is an area of increasing importance, especially with more devices and technologies being used in technical, industrial, and private sectors. These devices can be used to perform simple tasks, and gather data from the world around them. IoT devices are becoming increasingly popular for consumers, and the rise of smart-technology enabled homes has lead to an increase in the usage of low power devices.

Additionally, as the number of devices globally increases [2], the longevity of device manufacturers and developers relies on the use and reuse of previously or currently deployed devices. This would reduce overall production costs for new distributed systems, as well as increasing the material efficiency of in-place devices.

Distributed offloading systems cover a variety of different areas in existing literature. These areas include offloading technologies, automatic resource discovery, dynamic code deployment and execution, and resource negotiation.

2.1 Offloading Technologies

Offloading is an area that has been explored with increasing interest. Most of this focus is on the architectures and algorithms used when offloading computations. Mainly, methods that are used revolve around a Mobile Edge Computing (MEC) architecture, as explored in [16]. MEC architectures constitute an edge node communicating with the mobile client device, which can then communicate further with external cloud services. The edge node can communicate with other IoT devices, as well as cloud based servers. The main goal of this architecture is to offload from the mobile edge device, to reduce local power and computation requirements.

In the example presented in [11], applications for connecting multiple different mobile devices to the same network, This is a requirement for scalable dynamic MEC systems to be justified over the alternative of having one high powered device to complete otherwise offloaded computations. Additionally, [11] shows that such a system can complete complex tasks, such as Machine Learning algorithms, further justifying the use of a MEC system as a common place architecture.

[28] further looks at the joining of Small Cell Network (SCN) and MEC systems to allow for Non-Orthogonal Multiple Access (NOMA) algorithm to work on a similar architecture. In part, this is approaching the problem of bandwidth resource management; as more and more devices start using wireless communications, the usable range of network bands start to have a higher demand. Whilst this is a problem that is present for having a lot of devices working on the same SCN, the underlying architecture is focused on the application of MEC systems in the modern world. Whilst focusing on the bandwidth efficiency of this algorithm, the energy usage of the individual mobile device is taken into account within the bandwidth minimisation algorithm used.

In [25] and [14], new algorithms and architectures are proposed to solve highly specific problems. [25] explored the use of an edge device to compute energy harvesting techniques for IoT devices in a Fog architecture. This use of a edge device allows for IoT devices to interact with the network in a obfuscated manner: adding a layer of separation between the IoT device and the cloud system. This reduces the overall load on the cloud network, but as a trade-off, requires a MEC device in the end system. For this application, whilst each of the IoT devices exists in isolation, the application of a MEC device allows for computation to be done over the whole system, rather than on a single device. Similarly, [14] looked at computational offloading for IoT devices using a MEC system. However, this work begins to look at decentralised distributed systems as a mechanism for Offloading.

As seen in the explored literature, offloading has been a major focus when using IoT devices for data collection and decision making. However, at this stage, mobile devices are always offloading to a high powered device, in the form of a MEC device, or cloud services. Distribution of computations to local devices, following distributed programming techniques, is not applied to MEC architectures to create offloading systems.

2.2 Resource Discovery

In IoT networks, having flexibility in the deployment and usage of resources is important to allow for up-scaling the network. Mobile devices specifically have started to interact more with the other devices around them, especially when approaching IoT integrated mobile systems. As discussed in [27], some challenges with using dynamic resource discovery are recognising the existence of new nodes, allowing for the sharing and acquisition of knowledge, and predicting when new nodes will join the network for scheduling optimisation. Each of these challenges is important to plan for when creating a dynamic node network, to ensure that the requirements of timely communication and QoS are met.

The usage of low power devices in a dynamic node network also presents challenges, especially the energy usage of the communication method and protocol used between new and old devices. Bluetooth Low Energy (BLE) can be a useful connection strategy for IoT devices to avoid using a WiFi based connection protocol. [7] proposed sDiscovery; a discovery technology for use on low power devices using BLE communications. This method more efficiently uses energy than traditional device discovery, mainly through the use of BLE as the communication method. This focused on the energy usage of the system, which is an important aspect when using deployed IoT devices. Additionally, sDiscovery used an optimisation to the search protocol, allowing for the overall system to discover more effectively and efficiently.

Another method of reducing the discovery time and the energy usage can be achieved by clustering devices into "neighbourhoods". [10] discussed how this method could be used to achieve fast discovery in Device-to-Device (D2D) architectures. This method would allow for dynamic sizing of smaller "neighbourhoods" of devices, promoting a scalable device discovery architecture. Additionally, use of a decentralised system promotes the use of low power devices, without the requirement of a high power device to act as a routing node. However, testing is deployed using a simulation, without physically creating a network of low power devices.

Literature focusing on device discovery outlines the requirements of an IoT network discovery system; mainly the energy efficiency of the system, and the speed of the discovery and subsequent connection. Most work focuses on simulation results, rather than physical implementation with low power devices.

2.3 Dynamic Execution and Deployment

Distributed programming techniques are another method of Offloading computations from a Host device. By distributing computations out to multiple different devices, local energy can be conserved. QoS and Latency are still issues with this type of programming technique, however depending on the architecture and method, these can also be minimised. One approach that aides in distributed programming techniques is dynamic deployment. By dynamically allocating operations to individual devices, operations can be completed in a distributed manner whilst using each device effectively.

As proposed by [5], dynamic deployment of code using containers can optimise the usage of resources to complete a computation. In this case, Docker was used as a containerisation technology for a virtual cluster. Whilst this type of deployment can help to limit the run-time of certain operations, it does also require a higher level of computation to run multiple virtual machines in a cluster. Additionally, it further discusses creating sub-clusters for further breaking down operations into workable components.

Further, the concept of elastic clusters can be used, allowing devices to be dynamically allocated tasks, rather than either being in the cluster, or not in the cluster. As [8] proposed, containerisation of Virtual Elastic Clusters (VEC) can be used as a management system for virtual machines, in replacement of a Hypervisor. This use of elastic clusters focuses on reducing the overhead for individual virtual devices, allowing for more efficient VEC's to be used.

To demonstrate the applications of elastic clusters, [15] discusses using elastic clusters to coordinate wireless sensor networks using a Fireworks algorithm. This application of a distributed algorithm allows for stages of the algorithm to be completed with a faster convergence rate. However, the algorithm is still being deployed using VEC systems.

To further optimise elastic clusters, [20] proposed dynamic deployment of components for embedded systems. By outlining the existing components of a cluster, the components can be migrated to physical embedded devices. This allows for less computation on a host machine, and for embedded systems to do some calculations locally rather than only focusing computational power on data collection. This begins to approach the focus of this paper, as allowing for more computations to take place on low power devices would reduce the computational requirements for client devices.

At this stage, some literature is focused on the clustering and dynamic deployment of virtual machines. This still requires a higher powered network of devices to run the virtual machines. By using running computations on Idle low power devices, the computational requirement of all devices in the cluster can be reduced. QoS can be maintained through different distributed programming techniques.

2.4 Resource Negotiation

In addition to offloading, technologies for prioritising computations need to ensure that operation run-time and QoS meet given requirements through the process of offloading. Device negotiation approaches this problem by having shared agreements between the host (handling offloading resources) and client (generating offloading requests) devices. As [30] discussed, using a fog architecture to complete offloading requires low latency for operations such as augmented reality. To aid in this type of computation, edge devices may need incentives to complete the computation. As a result of the required motivation and low latency requirements, fast negotiation practices are required to allow for distributed offloading using the Fog network. Whilst this focus works well for decentralised High Powered Devices, for low powered devices such a system may have too much overhead, as incentive systems and negotiation practices require more communication and processing power.

Mesh networks, a type of fog architecture, can be used for individual Device-to-Device (D2D) negotiations. [17] took advantage of individual D2D negotiations for implementation of an energy monitoring and management system for an IoT network. This type of network allows for devices to reach an agreement based on their own computational load and specifications. A distributed network of IoT devices can make an agreement as to which devices will complete tasks required of the network. However, this method does take advantage of centralised task creation method. Having negotiations take place between individual devices, whilst a task list is created in a centralised manner leads to a semi-centralised system, requiring a control hub in addition to the smaller Low Powered Devices.

Further, QoS requirements and capability may not always match between the host network and the client device. As presented by [26], QoS can be negotiated between an end system and the offloading device. In the presented example [26], a simulation is used to model the communication between a mobile device with a IoT device. However, with a network of distributed devices, the overall specifications of the network may not be easily queried, and would need to be recalculated every time a device leaves or joins the network. This may not be suited for dynamic distributed systems, especially for low power devices.

2.5 Gap Analysis

This paper proposes an architecture to use idle low power devices for offloading distributed computations within a deployed IoT network. Exploration of offloading processes onto low power devices is a new area of research. Dynamic programming and deployment strategies can be used to aid in the creation of this architecture. The QoS, latency, run-time, energy usage, and materials usage are all concerns that must be taken into consideration when designing and testing the proposed architecture.

As can be seen from Table 1, the explored works do not cover a system which uses all areas of investigation. The aim of this paper is to outline how such a system would be designed and implemented.

The state-of-the-art analysis identifies the following areas that should be further investigated:

Table 1. Areas covered by existing literature

Literature	Sustainablity	Discovery	Offloading	Negotiation	Dynamic Deployment
[9]	✓		✓		
[10]		✓		✓	✓
[25]	✓		✓	✓	
[17]				✓	✓
[8]					✓
[1]	✓		✓		
[23]			✓		
[18]	✓		✓		
[24]			✓		✓
[19]	✓	✓			
[22]			✓	✓	✓

– Usage of idle computational resources to complete complex computations.
– Sustainability (material cost, energy use) of existing devices within a network.
– Offloading of computations onto a network of low power devices.
– Resource negotiation within offloading computations onto a network.
– Dynamic deployment of offloading computation.
– Low power devices with service discovery.

An IoT system that incorporates resource discovery, offloading of computations, and resource negotiations with dynamic deployment have not been identified or closely studied. Additionally, the area of sustainability within IoT networks and devices has not been thoroughly investigated, specifically with managing old or overpowered devices.

3 Localised Offloading Architecture for Distributed Horizontal Computations (LOADHoC)

The Localised Offloading Architecture for Distributed Horizontal Computations (LOADHoC) is the proposed solution for distributing computations among low power devices. This section aims to identify the core aims and requirements for the development of a prototype using the LOADHoC.

3.1 System Aims

To meet the aims of this work, an architecture is proposed for enabling the use of previously deployed devices for computing local offloading requests. The following aims have been defined to guide evaluations of localised distributed offloading with LOADHoC using the areas identified within Sect. 2.

- Aim 1: Creation of a dynamic system for decentralised offloading of computation.
  Using device discovery allows for ad hoc networks to be created per computation, meaning that devices can be added and removed from the network dynamically. Further, an important aspect of this architecture is to ensure that the latency and QoS are comparable to other offloading solutions.
- Aim 2: Designed for using low power devices for offloading.
  Understanding how networks of low power devices can be used for offloading is important for understanding whether such a system is achievable and useful.
- Aim 3: The system should be energy and resource efficient. Understanding the resources used for the architecture is important for identifying the sustainability and viability of the proposed architecture. This would involve analysing the energy use, latency, and physical material use.

3.2 System Requirements

Given the preceding System Aims, the final prototype design must adhere to the following functional requirements:

- Req 1: The proposed system must use low powered devices.
  Most offloading systems [13] currently use higher power devices. The aim of this paper is to take advantage of pre-existing low power devices within a network. As such, the prototype must use similar devices to emulate the practical application of the research.
- Req 2: The system must be able to perform offload-able computations.
  Offloading computations onto Low Powered devices allows for the prototype to be used to be useful manner, aligning with the aim of reuse of low power devices, and is key to meeting Aim 1.
- Req 3: The system must be decentralised for system requests and queries.
  In a practical situation, devices may be added or removed from the system, with the elimination of high power devices within the offloading process. As a result, the system must be decentralised to allow for each device to be removed from the network when needed.
- Req 4: The system must have measurable latency and QoS.
  Measuring the latency and QoS allows for the comparison of the proposed offloading architecture against performing the computation locally, and offloading to a cloud system.
- Req 5: The energy usage of the system must be able to be monitored, when completing operations.
  The energy usage must be monitored to ensure that the overall system is sustainable not only in terms of computational output per material used, but also in regard to energy usage.

Further, some non-functional requirements must be observed during the development of the LOADHoC.

– Req 6: The offloading computations must have comparable run-times to other offloading solutions.
  The run-time for offloading computations is important as the typical use of offloading is in real-time application. As a result, if the offloading is must slower, than the use of offloading on low power devices is not viable in a commercial application.
– Req 7: The devices must be able to dynamically disconnect and reconnect to the network.
  The system is designed for the use of pre-existing devices. However, when new devices join the network, or old devices leave the network, the system should be robust so that users do not need to manually add or remove devices from the offloading system.
– Req 8: The devices can be queried to display their computation.
  For diagnostic purposes, devices should be able to make available their current computation for testing. In a real world application, this decreases the security of the network, but making clear what computation is being completed allows for more careful and informative analysis of how effective the computation is, as well as analysis of the viability of the proposed system.

3.3 System Design

To meet these requirements, the LOADHoC is designed as a horizontally scaling system, with dynamic resource discovery. As can be seen in Fig. 1, each of the devices in the proposed network are of equal importance; with a distinction only being drawn when a client (the device requiring offloading) connects to the system. By having one of the devices act as a leader (the device handling the clients computation), this means that all of the devices in the network can receive instruction, and allows for no distribution of computation on the client device.

Each of the devices within the network requires two processes to function: these processes are the discovery and advertisement process, and the computation and request process.

The discovery and advertisement process takes advantage of mDNS addresses to advertise the device as a serviceable device. As can be seen in Fig. 2, this allows for other devices in the network to identify this device within the LOADHoC system. The process is then able to discover all other devices within the system, to keep track for when the new device is considered a Leader device.

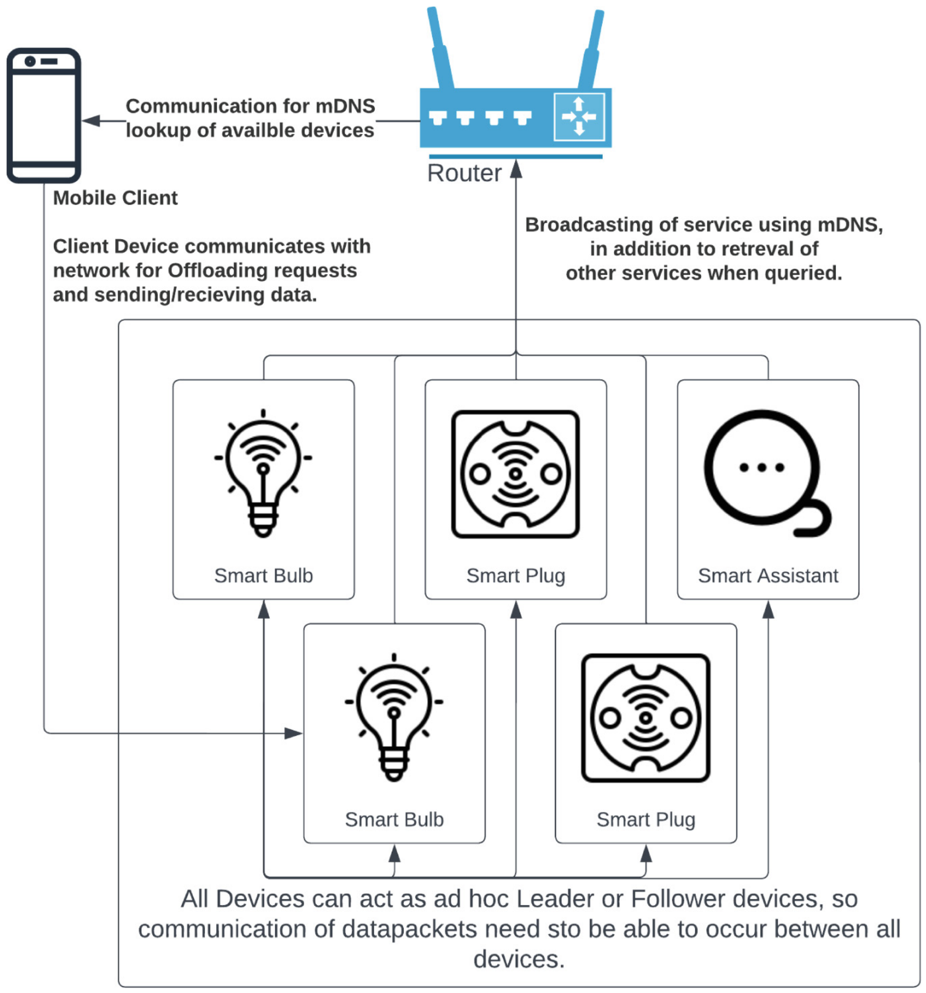
Communication for mDNS lookup of availble devices
Router
Mobile Client
Client Device communicates with network for Offloading requests and sending/recieving data.
Broadcasting of service using mDNS, in addition to retreval of other services when queried.
Smart Bulb
Smart Plug
Smart Assistant
Smart Bulb
Smart Plug
All Devices can act as ad hoc Leader or Follower devices, so communication of datapackets need sto be able to occur between all devices.

Fig. 1. Architecture Design of the LOADHoC

The computation and request process handles all requests for computation or information from the device, in addition to completing the computations that it receives. Importantly, this process must be able to act both as a leader or a follower device, as computations received directly from client devices will need to be split among all of the follower devices in the system, including the leader device.

The information from each individual device must be able to be queried by the system, as this allows for the leader device to check if a device is able to receive computation, as well as gain detailed information about the rest of the network that can then be used for allocation of computations. This, in addition to the previously outlined computations and dual functionality of the devices, requires the following methods must be present on all of the devices within the architecture:

– Info Request: Request the information of all of the devices within the network.

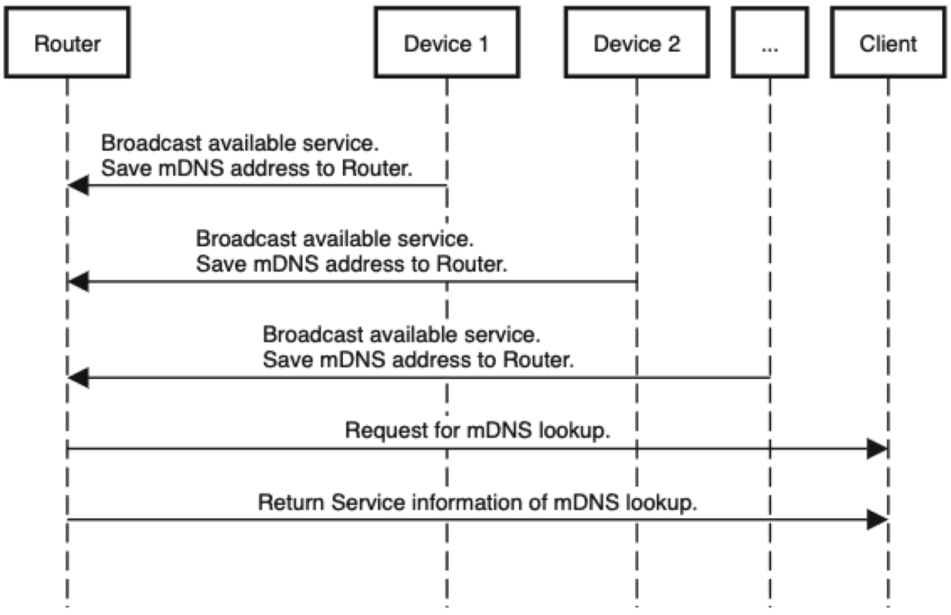
Router
Device 1
Device 2
...
Client
Broadcast available service.
Save mDNS address to Router.
Broadcast available service.
Save mDNS address to Router.
Broadcast available service.
Save mDNS address to Router.
Request for mDNS lookup.
Return Service information of mDNS lookup.

Fig. 2. Sequence for device advertisement method

This request would come from a Client device, and must be propagated through the network using the Info method.

- Info: Sends the info of the local device to the requesting device.
  This method is a direct response to the Info Request method. This endpoint will only be used by leader devices in the network requesting the device information.
- Ready Request: Request the ready status of all of the follower devices in the network.
  This method is similar to the Info Request method, and must return the current status of all of the follower devices in the network. This method would be requested by client device interacting with the system, to be further propagated through the remainder of the network.
- Ready: Sends the status of the local device to the requesting device.
  This method is a direct response to the Ready Request method. This endpoint should only be called by the leader devices, to compile information on the status of all devices for the Ready Request method.
- Computation Request: Receives the requested computation from the client device.
  This method is required on all devices as all devices must be able to act as a leader: able to accept computations from potential clients. This method focuses on filtering the type of computation requested, in addition to allocation of computation to other devices within the network.
- Computation: Receives computation from the leader device, completes and returns the result to the leader device.
  This endpoint will only be requested by the leader device in the context of the computation. This method receives portions of the overall computation, and returns the processed values.

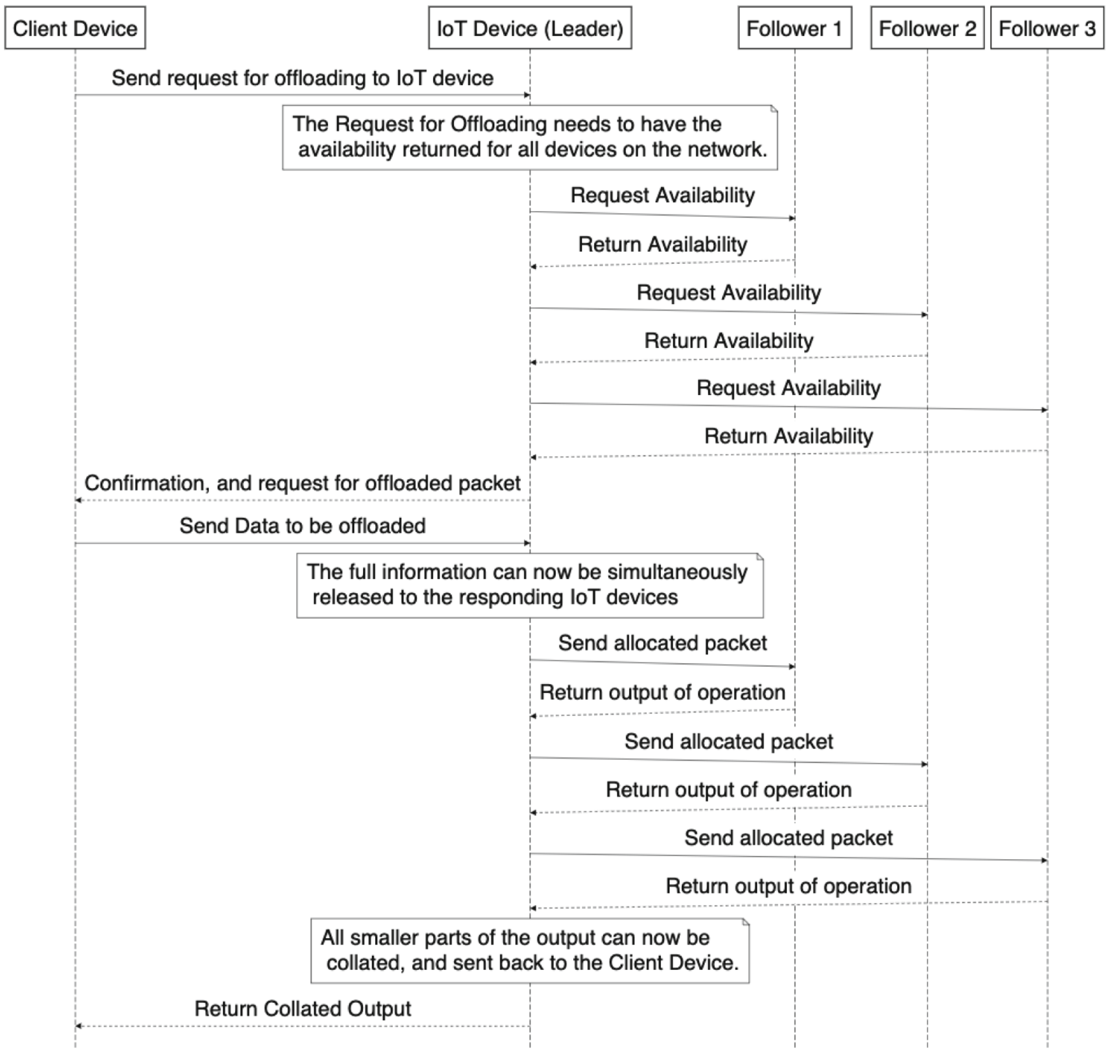
Client Device
IoT Device (Leader)
Follower 1
Follower 2
Follower 3
Send request for offloading to IoT device
The Request for Offloading needs to have the availability returned for all devices on the network.
Request Availability
Return Availability
Request Availability
Return Availability
Request Availability
Return Availability
Confirmation, and request for offloaded packet
Send Data to be offloaded
The full information can now be simultaneously released to the responding IoT devices
Send allocated packet
Return output of operation
Send allocated packet
Return output of operation
Send allocated packet
Return output of operation
All smaller parts of the output can now be collated, and sent back to the Client Device.
Return Collated Output

Fig. 3. Sequence Diagram for checking network status and completing computation.

The sequence of events for the status request system and computation request system are shown in Fig. 3. The status of the network, such as number of devices and processing ability of the devices, does not need to be retrieved before sending a request. The leader device should not wait for a portion of the computation to be completed before requesting another portion. This happens asynchronously, to allow for multiple devices to be working on computations simultaneously.

4 Experimental Evaluation

4.1 Experimental Setup

The experimental setup for all of the experiments use the LOADHoC prototype. The LOADHoC prototype is developed using a Restful API, taking advantage of Wireless Access Points (WAP) pre-existing on smart home devices. The use of devices already in place demonstrates the ability for the LOADHoC to operate without any additional components being added to the network. Smart home systems contain many of these types of devices, and would not require a dedicated device for functional offloading. Communication between the individual devices is conducted through API requests, with each individual device operating as an API.

The LOADHoC prototype was deployed to a cluster of 9 Raspberry Pi Zero Wireless devices (shown in Fig. 4) in addition to a Raspberry Pi 3b+. The Zero devices utilises a 1 GHz Broadcom BCM2835 single-core processor, with 512MB of RAM. Each board communicates with the rest of the network wirelessly using a Mediatek RT5370 (2.4 GHz B/G/N) chipset. The Zero boards run a Raspbian Lite OS, and the 3B+ a full Raspbian OS installation. The singular Raspberry Pi 3b+ includes a Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC processor with a 1.4 GHz clockspeed. This board additionally contains a wireless Cypress CYW43455 chipset. All devices were loaded with the heterogenous LOADHoC prototype, operating as leader and follower devices depending on which device receives the clients initial request.

When a client is interacting with the LOADHoCcluster, each device is able to operate as a routing device (splitting and sending computations to others within the system), and a worker device (completing requested computations). This means that if a request is sent to any of the devices, the computation will be split between the existing devices within the architecture.

Further, each of the devices within the LOADHoC is discovered automatically, with new devices able to participate in the load sharing protocol. This discovery is achieved using ZeroConf [3], an implementation of mDNS addresses designed for service and resource discovery. Devices are able to join the network at any time.

The system can support any offloaded computation, with testing occurring with matrix multiplication problems as the main measure, and accuracy of π also able to be completed with this implementation. Given the design, the system will be able to complete any offloaded computation given enough computational resources across all devices.

4.2 Experiment 1: System Validation

This experiment aims to assess the effectiveness of the proposed architecture. To analyse the effectiveness, the latency, run time, and QoS will be measured, and tested against non-distributed offloading, in addition to non-offloaded computation. This will be completed using the API implementation, to test the validity of the proposed architecture.

The setup for this experiment is focusing on the architecture, rather than the deployment to low power devices. In response to Aim 1, the latency, run time, and QoS will be individually measured.

The LOADHoC can be tested easily by deploying the API to three external devices, and creating a client to test the required metrics. The expected results from this experiment are that the run-time and latency are lowest when performing the computation locally, as the client device will be a high powered

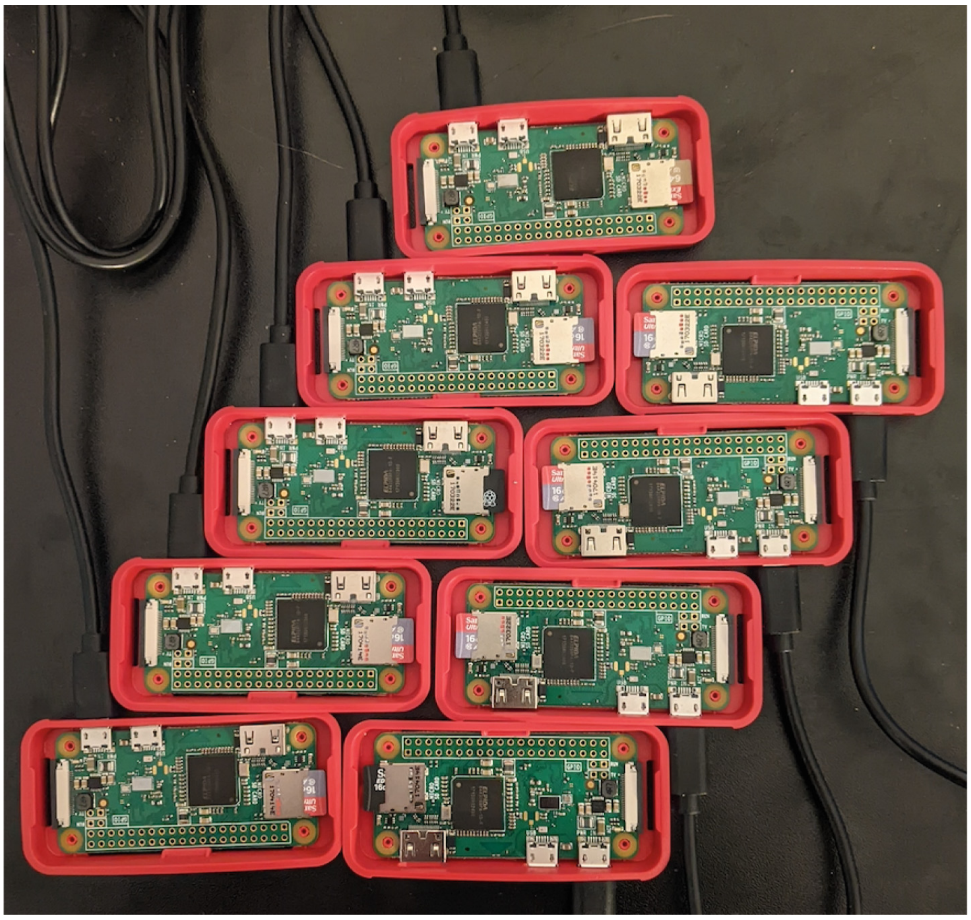

Fig. 4. 9 Raspberry Pi Zero W devices loaded with the LOADHoC Prototype

device. The non-distributed offloading and the distributed offloading systems are expected to be comparable in terms of latency and run-time, as the run-time and latency increase through the use of multiple devices will balance the reduction of sending the computation to multiple devices. However, as the number of devices increases, this balance will change, and the distributed offloading system is expected to outperform the non-distributed offloading method. Finally, the QoS should be the same among all three test-beds, which can be measured by testing the result matrix from each system against the other systems.

Figure 5 shows the time for each system to complete a matrix multiplication problem of the listed size. The tested methods are completion of the computation on the client device (Time locally), offloaded to one device (Time offloaded), and the time offloaded to the LOADHoC (Time offloaded with Distribution). The lowest run-time was achieved by completing the computation on the high-powered client device. This is expected, however this time may differ for mobile clients. Offloading the computation onto one local device produced acceptable run-times, but increases geometrically given the size of the matrices. This time may decrease or increase given the computational resources available to the single offloading device. The distributed system performed similarly to the single offloading system, demonstrating its viability when compared to traditional offloading systems.

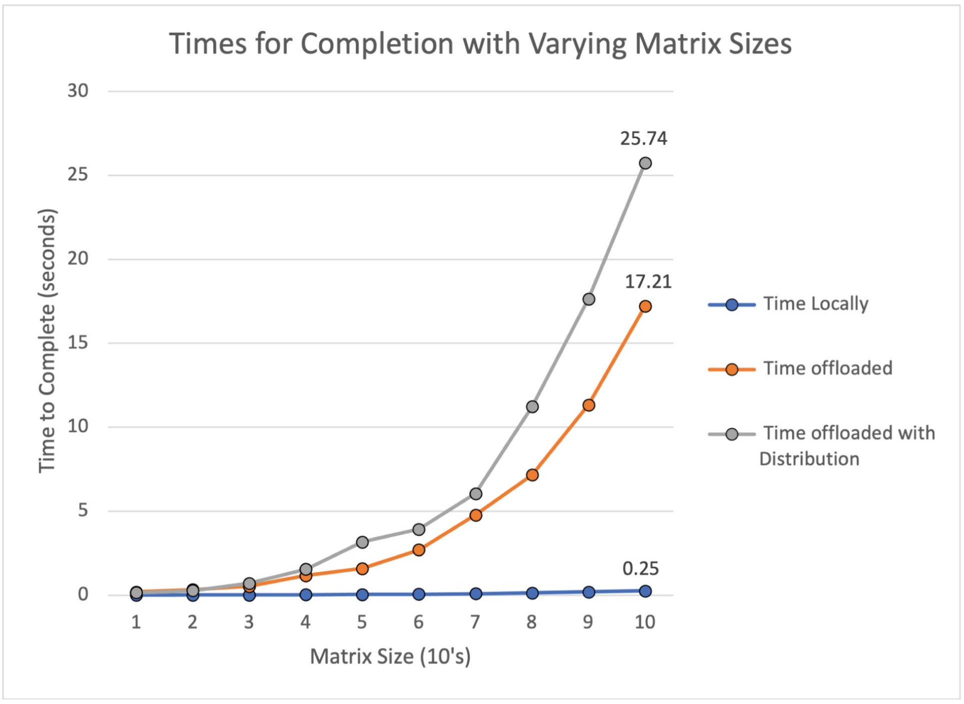

Fig. 5. Time-to-Complete for tested methods

Originally, the response time was much higher due to an implementation error. Each row of the second matrix was sent to devices using individual web requests, meaning the delay for each row was compounded. This inefficiency was fixed by removing all logging of data to the console, as well as changing the method to send multiple columns in the same request.

Additionally, it was shown that the distributed and non-distributed systems were comparable as the size of the matrices increased, with the distributed method taking longer. Experiment 2 tests the sustained efficiency of the system as more devices are added to the network. The non-distributed method performed better as the size of the matrices increased, however as the number of devices is increased, the response time should decrease for larger matrices.

When running computations, the individual power draw for one of the nodes in the LOADHoC can be measured, as shown in Fig. 6. This allows measuring the instantaneous power draw per device, which can give an energy usage rating to all of the devices in the network. As shown, the power draw does change when computations are being completed, by a constant amount. This method can also be used to visualise the computations being completed by the individual device.

Further, all three methods returned the same result matrix when presented with equivalent inputs for all tests. As the LOADHoC is producing an accurate result Matrix, the QoS has been preserved through the use of the API, with matrix sizes up to 100×100.

4.3 Experiment 2: System Scalability

This experiment aims to test the scalability of the LOADHoC. To analyse the architecture, the latency, run time, energy usage, and QoS is be measured for

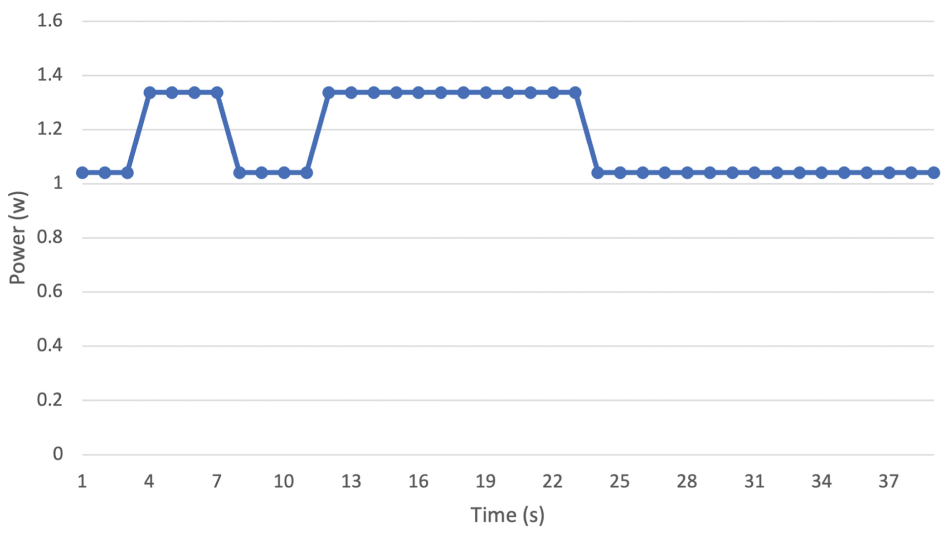

Fig. 6. Power usage of one Raspberry Pi within the system

computations that use up to 9 devices. These computations are distributed through the LOADHoC, using a standard 100 by 100 matrix.

The results of this experiment outline the scalability of the LOADHoC prototype for small scale calculations on medium size networks. The LOADHoC implementation can handle a large number of devices within the network, limited only by the unique number of IP addresses on the network. Testing the run-time of requests within the system as the number of devices increases allows for understanding the minimum number of devices required to complete computations faster than other traditional offloading and local calculation methods.

The LOADHoC prototype has been developed to allow for the number of devices within the system to dynamically change. For this experiment, 9 devices, including the Leader device, are connected within the system. In testing the LOADHoC, two 100×100 matrices are multiplied together, as requested by the client device.

The latency and QoS are measured by the Client device, using the amount of time that it takes to get the result from sending the computation. Additionally, the energy usage of the overall system will be monitored, using one energy collection device to measure the usage of the entire system, and one to measure the usage of an individual device.

This experiment allows the system to be tested on larger networks; testing its run-time and energy usage, compared against performing the computation on a smaller number of devices. Additionally, the method that the LOADHoC uses to handle distributed computations can also be evaluated.

Figure 7 details the time-to-complete the multiplication of two 80×80 matrices. As can be seen, this time increases as the number of devices increases. The lowest run time occurs with one device, as communication only needs to occur between the client and leader device. As the number of devices increases, the run-time also increases due to the number of computation requests that must be sent to each of the follower devices.

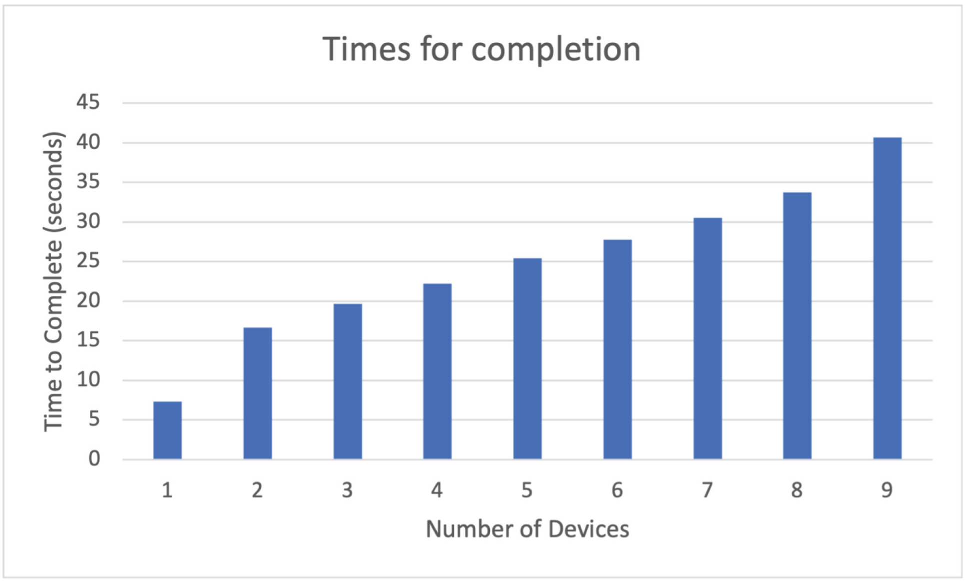

Fig. 7. Time to complete with given number of devices

Figure 8 displays the energy usage of the network when offloading the multiplication of two 100 × 100 matrices across 9 devices. As more devices receive their computational load, the energy demand of the system increases to accommodate the extra processing power required by the worker devices. The baseline energy usage of these devices remains unchanged before being added to the LOADHoC. This means only the increase of energy use during a computation must be considered when analysing the energy usage of the system.

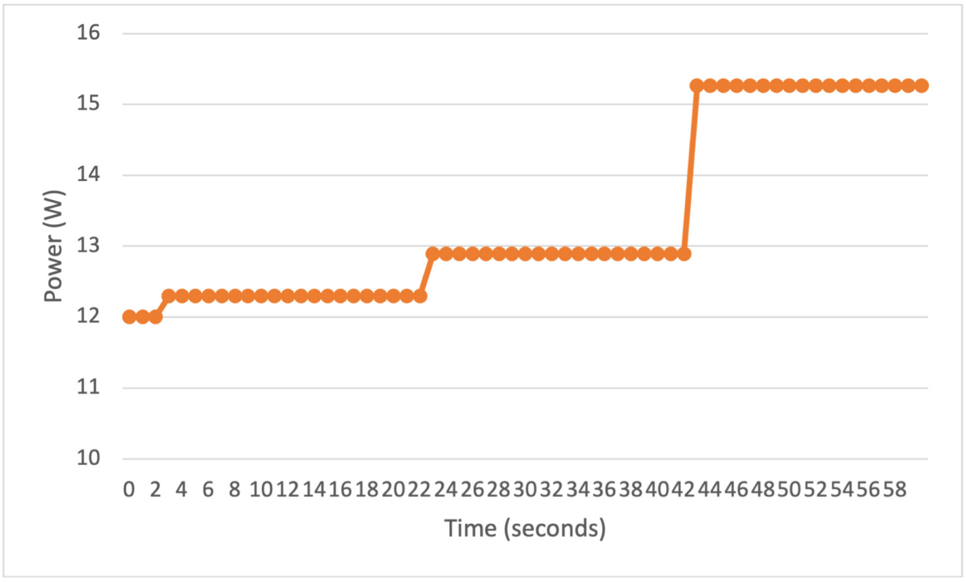

Fig. 8. Energy for 100 × 100 multiplication across 9 devices

In this case, offloading to one device is seen to be the most optimal solution for completion time and energy use. However, the LOADHoC implementation is currently implemented on high power devices, and as such, are able to complete more complex operations by themselves. These devices are able to complete computations effectively by themselves. The main time-sink is the sending of messages between the leader and follower devices.

4.4 Experiment 3: Computation Flexibility

The aim of this experiment is to demonstrate and test the system using another distributed computation. Implementing a second computation algorithm demonstrates the suitability of the LOADHoCfor dynamic deployment methods. This flexibility in the computations that can be completed is integral for the system having wider applications.

For this experiment, a estimation of π will be implemented as a distributed computation across the system. The accuracy will be tested to ensure that the result that is achieved by the system matches a locally calculated value.

The LOADHoC system has been enhanced to include the calculation of Pi. In order to implement the π calculation, the system must have the distributed computation method on all of the devices. To demonstrate the LOADHoC's flexibility, the overall system architecture needed to remain functional, as well as maintaining the matrix multiplication functionality.

The π estimation method used within this experiment is the Nilakantha infinite series [4]. The Nilakantha series uses the following equation:

$$\pi \approx 3 + 4 \sum_{n=0}^{\infty} \frac{-1^n}{(2n+2)(2n+3)(2n+4)} \tag{1}$$

As implemented within the prototype, the maximum value of n is the accuracy of the computation, which can be specified by the client device when retrieving the value. The values of n can then be distributed to each of the devices, the calculation of each section distributed to the follower devices, then added together on the leader device.

Listing 1.1. Code for estimating Pi locally

```
function calcPi(accuracy):
    result = 3
    for( i in range(0, accuracy / 2)):
        n = i * 2
        m = n + 1
        result+= (1/((2n+2)(2n+3)(2n+4))) - (1/((2m+2)(2m
            +3)(2m+4)))
    return result
```

Listing 1.1 highlights how this code can be used for estimating π on a local device. By giving individual devices differing values of i, the function can be distributed across the for-loop.

This experiment outlines the ability for the LOADHoC to be adapted for more distributed programming applications. The adaptation the prototype requires identification of the requested computation, distribution and collection of the computation, and calculation on the follower devices.

The testing value of π for this experiment is as follows:

$$\pi \approx 3.14159265358979323846264338327 \tag{2}$$

Table 2. Values retrieved from distributed calculation of π

Value of n	Value of π Obtained	No. of accurate decimals
$n = 10$	3.222821622821623	0
$n = 100$	3.141620854093511	3
$n = 1000$	3.1415927233510117	6
$n = 10000$	3.141592653593063	10
$n = 100000$	3.1415926535897905	14
$n = 1000000$	3.1415926535897932	16

The values obtained range in accuracy given the depth of the π calculation. The values within Table 2 were retrieved by requests to the LOADHoC implementation. Further levels of accuracy can be obtained given larger values of n. However, a limitation of the current system is the implementation, and the number of decimal places available. The LOADHoC implementation is currently developed in JavaScript, and limits the number of decimal places available to be received by the client. This is largely a problem that exists with this calculation and this implementation, so this does not effect the validity of the overall LOADHoC implementation.

This experiment demonstrates the flexibility of the LOADHoC architecture. The prototype is able to handle different distributed computations, which are only limited by the systems programmed on the networked devices.

4.5 Experiment 4: Resource Allocation

This experiment aims to demonstrate the implementation of different types of allocation methods for resources. Given the different calculations achievable, and differing devices in the architecture, systems for allocation different sized loads to different devices.

This experiment will record the timing of each of the different allocation methods (including calculation), as well as the energy use of the system using these methods. The output of the different allocation methods can also be tested, to ensure the QoS of the overall system is maintained with differing resource allocation methods.

The previously outlined LOADHoC prototype is used for this experiment. In addition to the calculations, a resource allocation method is implemented for allocating the resources between those available in the network. Within the LOADHoC implementation, four allocation methods are present. These allocation methods are:

– “none”: Default allocation method, which splits the computation over the requested number of devices.
– “even”: Allocates the computation to every second node, relative to their position in the leader device list.
– “ready”: Similar to the default allocation method, allocates computation only to ready devices. Devices are considered ready if they are not currently completing another operation.
– “geo”: Allocates multiple computation regions to the same device; increasing by one portion of the overall computation as each device is allocated its computations.

Each of these methods must be set within the leader device local data structure. Once set, all calculations using the altered device as leader will use the specified allocation method. Importantly, different methods can be defined for differing leader devices. This means multiple users are able to define different allocation methods while using the system simultaneously.

This experiment demonstrates that allocation methods can be created and used for the LOADHoC architecture. This implementation uses four different methods, while showing the effective run-time of the same calculation across all methods. Further, more complex allocation methods can be used effectively with this system, using the data made available by each of the devices.

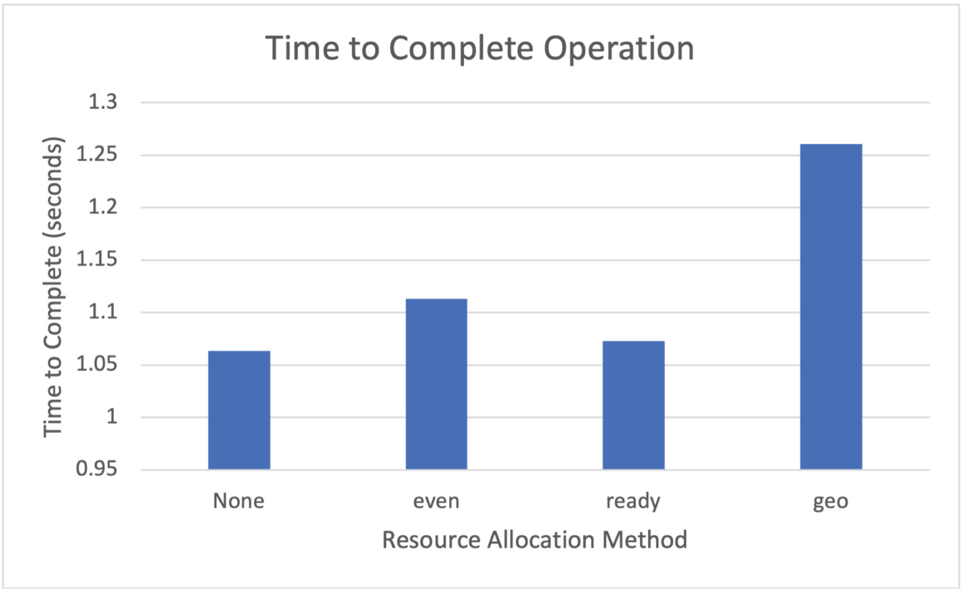

Fig. 9. Time to complete with different allocation methods

As can be seen in Fig. 9, the time to complete the different allocation methods was comparable between all methods, other than the "geo" method. This is mainly due to the number of requests being sent to the same device. Increasing the load on an individual device, when all of the devices are of equal computing and processing power leads to an increase in the total completion time. The Geo method was unable to resolve as the request was taking too long to be fulfilled. This is because the device that was loaded with the most computation was similar in power to the remainder of the devices, and was required to compute many different parts of the computation simultaneously.

This experiment outlines that different allocation methods are possible, in addition to querying information from the devices to inform this allocation. The LOADHoC architecture is able to take advantage of more advanced allocation methods, and only requires the leader device to manage the allocation of computation within the network.

4.6 Experiment 5: Robustness

The aim of this experiment is to evaluate the failure recovery methods for the LOADHoC architecture. The LOADHoC implementation is able to freely add, remove, and drop devices. This allows individual devices to cease normal operation, whilst maintaining the integrity of the overall architecture and network.

The experiment records the QoS of the given computation after a device is forcefully removed from the network. By simulating a device being removed from the system in the middle of a requested computation, the way that the LOADHoC architecture can handle errors can be observed.

This experiment uses the previously outlined LOADHoC implementation. Each of the nodes in the network uses two separate processes: advertisement and discovery, and a computation process. By shutting down the computation process of an individual device, a connection or computation error can be simulated. Importantly, this simulates when a device that is actively on the network is not responding in an expected manner.

When sending a request to a new device, if the request fails, then the same request is sent to a new device listed on the Leader devices device list.

Testing the system in this way tests the LOADHoC's ability to support recovery and retention of computation when individual devices fail. As the client only communicates with the Leader device, the follower devices are able to drop in and out of the network without effecting the ability for the system to function.

Multiple devices can be removed from the network. This will effect the overall run-time to comparable times to limiting the number of devices a computation can use. This test will look at the QoS of the LOADHoC implementation when devices are removed.

This experiment focuses more on how the architecture can be implemented. By only relying on the Leader device to function, a system running the LOADHoC architecture can be scaled horizontally without negatively effecting the network. Additionally, parts of the network can be removed without effecting the integrity of the remaining network.

Overall, as devices were forcefully removed from the system, the LOADHoC implementation was able to cope with the removal, and reallocated the specific portion of the computation to an active device. This was done by querying all of the devices in the leaders device list, which includes itself. However, this did effect the total run-time of the system, in a similar manner to the results of the experiments presented in Subsect. 4.3. However, when removing the leader device in the middle of a computation, the result will not be returned to the client. The leader device is determined by which device receives the request from the client device. As such, other devices in the network can be queried, so re-sending the computation request from the client side will result in the computation being completed, maintaining the system fidelity. This removal does not effect other computations using the removed device as a worker.

These results demonstrate that the overall architecture is able to cope with individual devices being removed from the network. In applications where network security and stability are not ideal, the system will remain to function in an optimal condition given the client remains connected to the leader device.

5 Discussion

Addressing the aims of the LOADHoC implementation, as defined in Subsect. 3.1, the experimental results can be split into three areas.

5.1 Creation of a Dynamic System for Decentralised Offloading

Through the development of the LOADHoC, the requirements outlined in Subsect. 3.2 were used to allow the overall design to achieve dynamic centralised offloading. Subsection 4.2 outlined the use of the system using matrix multiplication as a test problem. In addition, Subsect. 4.4 focuses on a calculation of π, highlighting the ability for different computations to take place using the system.

By taking advantage of multiple low powered devices, many different types of computations can be completed, limited only by the time constraints placed on the computation. Given the results achieved in Subsect. 4.2, computation runtime could feasibly fall outside the request timeout set by the client device. This can be tweaked, by changing the method of request delivery and sending. Use of a message queue would allow for dynamic Time-To-Live (TTL) for requests and results. The LOADHoC is able to cope with multiple different types of resource management, as demonstrated within Subsect. 4.5. The LOADHoC is designed to be dynamic in both computation and resource allocation method, allowing for further implementation in real-world dynamic applications where local offloading is required.

5.2 Design for Implementation on Low Power Devices

The overall design of the LOADHoC allows for use of low power devices to run and work on the network. This was achieved using a RESTful API system as a demonstration, allowing definition of methods and endpoints, as outlined in Subsect. 3.3. Whilst experiments were run on high powered devices, the network connectivity was the main difference in run-time when compared to a traditional offloading system. As the LOADHoC needed to communicate with several devices, offloading to a dedicated device was substantially faster, as seen in Subsect. 4.2. This time difference was also outlined in Subsect. 4.3, as use of 9 devices produced the largest run-time. In addition to time delay through communication within the network, the use of high powered devices splitting processing power among multiple passive processes can also reduce the amount of available resources for computation. This is highlighted by Subsect. 4.2, where even offloading to a single high powered device significantly increased the run-time of the computation. Distribution of the computation across the network decreased this time inefficiency, while still taking longer to complete the computation. Low power devices can also be easily fault or disconnect from networks in real-world applications. As seen in Subsect. 4.6, the LOADHoC can cope with device reliability issues, allowing for QoS to be maintained for error-sensitive computations. The LOADHoC design lends itself to use on low powered devices, through simple implementation design and robust handling of device and communication errors.

5.3 Energy and Resource Usage and Efficiency

The LOADHoC implementation was tested on run-time, as well as energy and resource usage. The main outcome from the implementation is that the architecture would work on low power devices, as discussed in Subsect. 5.2. Additionally, it can be seen in Subsect. 4.2 that the energy usage of the system is comparable to having the devices passively in the network. Given the implementation of the LOADHoC onto pre-existing devices, the overall energy and resource consumption is minimal compared to having a dedicated device for computation. The energy from each of the devices does add up, as can be seen in Subsect. 4.3, but still remains less than the overall passive power of the devices within the network. The current power readings from the network are a reflection of the power requirements for an implementation of the LOADHoC using low power devices. However, the values may differ, effecting the overall efficiency of the networks energy use. When analysing resource use, taking advantage of pre-existing devices ultimately saves on cost, in addition to resource and material use.

6 Conclusion and Future Work

Through this work, horizontal distribution of computation has proven to be a viable method of distributing computation. Whilst the overall time constraints and energy usage are higher than the use of a single high powered device, the computations being run on each individual device can be implemented on lower power devices for in-place computation; without the addition of new devices into a network. This decreases the overall price of using a localised offloading network, at the cost of time-to-complete. Whilst the energy efficiency is lower than using a single high powered device, as discussed in Sect. 5, the increased energy usage is minimal when compared to a new high powered device on the network.

Future work focuses on the implementation of the LOADHoC onto low powered devices, testing the run-time and resource usage to determine viability in real-world resource constrained networks. In addition, different methods of communication can be tested, such as a message queue system.

References

1. Agarwal, P., Kumar, G.V.V.R., Agarwal, P.: IoT based framework for smart campus: COVID-19 readiness. In: 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), pp. 539–542 (2020). https://doi.org/10.1109/WorldS450073.2020.9210382
2. Benazzouz, Y., Munilla, C., Gúnalp, O., Gallissot, M., Gúrgen, L.: Sharing user IoT devices in the cloud. In: 2014 IEEE World Forum on Internet of Things (WF-IoT), pp. 373–374 (2014). https://doi.org/10.1109/WF-IoT.2014.6803193
3. Bohnenkamp, H., van der Stok, P., Hermanns, H., Vaandrager, F.: Cost-optimization of the ipv4 zeroconf protocol. In: 2003 International Conference on Dependable Systems and Networks, 2003. Proceedings, pp. 531–540 (2003). https://doi.org/10.1109/DSN.2003.1209963
4. Brink, D.: Nilakantha's accelerated series for pi. Acta Arithmetica **171**, 293–308 (2015). https://doi.org/10.4064/aa171-4-1
5. Cao, Y., Wang, H., He, S.: Dynamic deployment model for large-scale compute-intensive clusters. In: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 586–591 (2020). https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162887
6. Caprolu, M., Di Pietro, R., Lombardi, F., Raponi, S.: Edge computing perspectives: architectures, technologies, and open security issues. In: 2019 IEEE International Conference on Edge Computing (EDGE), pp. 116–123 (2019). https://doi.org/10.1109/EDGE.2019.00035
7. Chen, B.R., Cheng, S.M., Lin, J.J.: Energy-efficient BLE device discovery for internet of things. In: 2017 Fifth International Symposium on Computing and Networking (CANDAR), pp. 75–79 (2017). https://doi.org/10.1109/CANDAR.2017.95
8. de Alfonso, C., Calatrava, A., Moltó, G.: Container-based virtual elastic clusters. J. Syst. Softw. **127**, 1–11 (2017). https://doi.org/10.1016/j.jss.2017.01.007, https://www.sciencedirect.com/science/article/pii/S0164121217300146
9. Hou, W., Li, W., Guo, L., Sun, Y., Cai, X.: Recycling edge devices in sustainable internet of things networks. IEEE Internet Things J. **4**(5), 1696–1706 (2017). https://doi.org/10.1109/JIOT.2017.2727098
10. Huang, P.K., Qi, E., Park, M., Stephens, A.: Energy efficient and scalable device-to-device discovery protocol with fast discovery. In: 2013 IEEE International Workshop of Internet-of-Things Networking and Control (IoT-NC), pp. 1–9 (2013). https://doi.org/10.1109/IoT-NC.2013.6694047

11. Huang, X., Leng, S., Maharjan, S., Zhang, Y.: Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks. IEEE Trans. Veh. Technol. **70**(9), 9282–9293 (2021). https://doi.org/10.1109/TVT.2021.3096928
12. Janeczek, K.: Composite materials for printed electronics in internet of things applications. Bull. Mater. Sci. **43**(1) (2020). https://doi.org/10.1007/s12034-020-02101-x
13. Jha, S.C., Gupta, M., Koc, A.T., Vannithamby, R.: On the impact of small cell discovery mechanisms on device power consumption over LTE networks. In: 2013 First International Black Sea Conference on Communications and Networking (BlackSeaCom), pp. 116–120 (2013). https://doi.org/10.1109/BlackSeaCom.2013.6623393
14. Lei, L., Xu, H., Xiong, X., Zheng, K., Xiang, W.: Joint computation offloading and multiuser scheduling using approximate dynamic programming in NB-IoT edge computing system. IEEE Internet Things J. **6**(3), 5345–5362 (2019). https://doi.org/10.1109/JIOT.2019.2900550
15. Liu, X., Zhang, X., Zhu, Q.: Enhanced fireworks algorithm for dynamic deployment of wireless sensor networks. In: 2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST), pp. 161–165 (2017). https://doi.org/10.1109/ICFST.2017.8210494
16. Mach, P., Becvar, Z.: Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. **19**(3), 1628–1656 (2017). https://doi.org/10.1109/COMST.2017.2682318
17. Mannan, A., Bader, M., Hijawi, U., Gastli, A., Hamila, R.: Mesh network-based device-to-device negotiation system for energy management. In: 2021 18th International Multi-Conference on Systems, Signals Devices (SSD), pp. 1091–1098 (2021). https://doi.org/10.1109/SSD52085.2021.9429379
18. Mylonas, G., Amaxilatis, D., Chatzigiannakis, I., Anagnostopoulos, A., Paganelli, F.: Enabling sustainability and energy awareness in schools based on IoT and real-world data. IEEE Pervasive Comput. **17**(4), 53–63 (2018). https://doi.org/10.1109/MPRV.2018.2873855
19. Narottama, B., et al.: Base station energy efficiency of d2d device discovery. In: 2017 International Conference on Control, Automation and Information Sciences (ICCAIS), pp. 257–261 (2017). https://doi.org/10.1109/ICCAIS.2017.8217586
20. Petricic, A.: Predictable dynamic deployment of components in embedded systems. In: 2011 33rd International Conference on Software Engineering (ICSE), pp. 1128–1129 (2011). https://doi.org/10.1145/1985793.1986015
21. Qian, H., Andresen, D.: Extending mobile device's battery life by offloading computation to cloud. In: 2015 2nd ACM International Conference on Mobile Software Engineering and Systems, pp. 150–151 (2015). https://doi.org/10.1109/MobileSoft.2015.39
22. Song, B., Shihong, G., Xiaolong, Q.: Application of blackboard model based multiagent negotiation system in the distributed design of device-layer network. In: The 27th Chinese Control and Decision Conference (2015 CCDC), pp. 6005–6010 (2015). https://doi.org/10.1109/CCDC.2015.7161886
23. Srividhya, S., Sankaranarayanan, S.: IoT-fog enabled framework for forest fire management system. In: 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), pp. 273–276 (2020). https://doi.org/10.1109/WorldS450073.2020.9210328

24. Sun, H., Wang, S., Zhou, F., Yin, L., Liu, M.: Dynamic deployment and scheduling strategy for dual-service pooling based hierarchical cloud service system in intelligent buildings. IEEE Trans. Cloud Comput. 1 (2021). https://doi.org/10.1109/TCC.2021.3078795
25. Tang, Q., Xie, R., Yu, F.R., Huang, T., Liu, Y.: Decentralized computation offloading in IoT fog computing system with energy harvesting: a dec-POMDP approach. IEEE Internet Things J. **7**(6), 4898–4911 (2020). https://doi.org/10.1109/JIOT.2020.2971323
26. Udoh, I., Kotonya, G.: A dynamic QoS negotiation framework for IoT services. In: 2019 IEEE Global Conference on Internet of Things (GCIoT), pp. 1–7 (2019). https://doi.org/10.1109/GCIoT47977.2019.9058418
27. Valarmathi, M., Sumathi, L., Deepika, G.: A survey on node discovery in mobile internet of things(IoT) scenarios. In: 2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS), vol. 01, pp. 1–5 (2016). https://doi.org/10.1109/ICACCS.2016.7586400
28. Yang, L., Guo, S., Yi, L., Wang, Q., Yang, Y.: NOSCM: a novel offloading strategy for NOMA-enabled hierarchical small cell mobile-edge computing. IEEE Internet Things J. **8**(10), 8107–8118 (2021). https://doi.org/10.1109/JIOT.2020.3042318
29. Yu-Jie, S., Hui, W., Cheng-Xiang, Z.: Balanced computing offloading for selfish IoT devices in fog computing. IEEE Access **10**, 30890–30898 (2022). https://doi.org/10.1109/ACCESS.2022.3160198
30. Zeng, M., Li, Y., Zhang, K., Waqas, M., Jin, D.: Incentive mechanism design for computation offloading in heterogeneous fog computing: a contract-based approach. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6 (2018). https://doi.org/10.1109/ICC.2018.8422684
31. Zhang, Z., Li, S.: A survey of computational offloading in mobile cloud computing. In: 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), pp. 81–82 (2016). https://doi.org/10.1109/MobileCloud.2016.15