information

Article

S.O.V.O.R.A.: A Distributed Wireless Operating System

Henry Zárate Ceballos [1,*,†,‡] and Jorge Eduardo Ortiz Triviño [2,‡]

1 Department of Researcher Systems and Industrial, Engineering Faculty, Universidad Nacional de Colombia, Bogotá 110011, Colombia
2 Department of Associate Professor Systems and Industrial, Engineering Faculty, Universidad Nacional de Colombia, Bogotá 110011, Colombia; jeortizt@unal.edu.co
* Correspondence: hzaratec@unal.edu.co; Tel.: +57-317-3824-453
† Current address: Carrera 45 No. 26-80, Bogotá 11011, Colombia.
‡ These authors contributed equally to this work.

Received: 21 October 2020; Accepted: 7 December 2020; Published: 14 December 2020

Abstract: Due to the growth of users and linked devices in networks, there is an emerging need for dynamic solutions to control and manage computing and network resources. This document proposes a Distributed Wireless Operative System on a Mobile Ad-hoc Network (MANET) to manage and control computing resources in relation to several virtual resources linked in a wireless network. This prototype has two elements: a local agent that works on each physical node to manage the computing resources (e.g., virtual resources and distributed applications) and an orchestrator agent that monitors, manages, and deploys policies on each physical node. These elements arrange the local and global computing resources to provide a quality service to the users of the Ad-hoc cluster. The proposed S.O.V.O.R.A. model (Operating Virtualized System oriented to Ad-hoc networks) defines primitives, commands, virtual structures, and modules to operate as a distributed wireless operating system.

Keywords: distributed system; operative system; local agent; orchestrator; container; MANET

1. Introduction

The inclusion of communications on computer systems has allowed for new dynamics between devices and users, the deployment of new services, and the demand for new features. The constant evolution of hardware and communication systems enables new kinds of technologies that increase the productivity of traditional systems such as the Internet, education, surveillance, monitoring systems, networks, and others.

Therefore, there are more wireless technologies embedded in a mobile device and supporting different cellular networks such as 2G (GSM CSD, GPRS); 3G (UMTS/HSDPA); 4G (WiMAX, LTE, LTE-A (HSPA + LTE)); and 5G technologies. These rely on a central base station and a set of cells to provide services and coverage. However, these systems are prepared to support short-range communications that have rapidly evolved by reducing the size and increasing the computing capacity of microchips, allowing for overlapping networks within the same device. Two technologies are outstanding in that evolution: Bluetooth and IEEE802.11.x. When using wireless mobile devices, it is possible to build networks known as MANET (Mobile Ad-hoc Networks), which can be configured autonomously, do not require centralized control, and can automatically recover in case of failure. An alternative to exploit the features offered by MANET is to use mobile clouds [1] as a platform to sharing resources in a cluster lead to take advantage to MANET features.

The massive number of users and services linked together brings new challenges due to the dynamic and stochastic behavior of distributed systems such as the Internet. Users consume applications and computing resources that are eventually shared on the network. The newly developed architectures provide users with Information Technologies services (IT), applications, platforms, information, monitoring, control systems, etc., to manage services and administer resources between the edge and the core network.

Currently, the devices and systems deployed on the edge layer can be classified in three types [2]: Mobile Edge Computing (MEC), Fog Computing (FC) and Cloudlet Computing (CC). The MEC includes interactions with cellular networks and offers some cloud services in the cellular cell; the FC has a computing layer before the Cloud layer to store and process data; finally, the CC is deployed on dedicated devices with robust computing capacities, commonly called micro data centers.

The main contribution of this paper is the design and implementation of a distributed wireless operating system, dedicated to the dynamic management of computing resource in MANET devices [3]. The goal is to find a solution to dynamically (re-)organize computation tasks on available network resources and node status. The operating system uses two elements to create and deploy virtual resources: abstraction and the primitives, which are managed by the Local Agent (LA) that resides as an instance on each physical node and the Orchestrator (OR) that work on one or more nodes to monitor and deploy policies on the distributed operating system.

This paper is organized as follows: Section 2 presents a review of the existing solutions similar to the one proposed here. Section 3 introduces the reference models adopted in the debate for the system architecture and application workload. Section 4 describes the implemented prototype and the experimental demonstration of the proposed solution, showing its effectiveness. Finally, Section 5 presents the final observation and future work.

2. Wireless Distributed Systems

Distributed systems were developed to manage resources in different locations and computing devices through network links. Today, services and systems need mobility as an extra feature in addition to cloud computing architectures that mitigate the problem of processing and latency. Since issues related to latency and computational load can arise, it is essential to manage workflows between applications and final devices. In this context emerges the concept of Edge Clouds [4,5], which have been proposed to improve the Quality of Experience (QoE). In this paper, we use the Wireless Distributed Computing (WDC) [6] concept as a framework to develop and validate the S.O.V.O.R.A. Model proposed. WDC exploits wireless connectivity to share processing-intensive tasks among multiple devices.

2.1. Network Management

To manage a network, wired or wireless, a lot of protocols, architectures, tiers and abstraction levels exist and are used to deliver services to the users, and this condition made the network management and control complex. The SDN proposes to decouple the control plane from the data plane on a network. With this architecture, the routers and switches became logical elements of the network provided by an external entity called Network Operation System (NOS). This controller allows for the generation of abstractions that deploy orchestrated services at logical and virtual levels allowing automation and control.

In this sense, NFVs are in charge of generating Virtualized Network Functions (VNF) by separating the network functions from the hardware and offering it through virtualized services or in general-purpose servers [7–9]. This deployment of functions requires less hardware but includes more software abstractions for better traffic management.

2.2. Distributed Architectures

For this case study, the enabling technologies are wireless communication, embedded system with high multiprocessing capabilities and the virtualization techniques, which allows for the development of a virtual system on a wireless cluster or network in order to provide systems with real time data capture, processing and dissemination, information sharing between users and nodes, robust process control and resource management. In order to reach this goal, it is necessary to identify and analyze the distributed architectures that are used nowadays.

As classical distributed architecture Cloud computing allows for the modeling of ubiquitous or on-demand network access and resource sharing capabilities to be quickly and automatically provisioned. In this sense, customers only pay for what they use. The service models in cloud computing are divided into three categories: Software as a Service (SaaS), Platform as a Service (SaaS), and Infrastructure as a Service (IaaS). Orchestration services involve the dynamic deployment, administration, and maintenance of services on the mentioned platforms, according to the needs of customer's. However, on wireless networks, distributed architectures such as MANETS, Fog Computing, and IoT systems exist as study cases [10,11].

2.2.1. MANET Mobile Ad Hoc Network

A MANET [12] is a set of wireless mobile devices (nodes), which can form a network without the support of any infrastructure or centralized control. MANETs are multi-hop networks in which a packet is sent from a source to its destination and must cross a path formed by two or more hops (nodes).

Therefore, every node has a dual behavior in a MANET as a router and host. MANETs are autonomous networks that can determine their configuration parameters and recover in the case of failure by themselves. Further, they are implemented to set up communications for specialized applications where there is no pre-existing infrastructure or the one available is not the most suitable for the operation's needs.

There can be two types of MANETs: Wireless Mesh Networks (WMNs) [13] and Wireless Sensor Networks (WSNs) [14]. As is common in MANETs, a node can function as a router and as a host, while the WMN nodes are classified as mesh routers or mesh nodes. On the one hand, mesh routers have minimal mobility, provide access for normal nodes and mesh nodes, and can communicate with other mesh routers; besides, they are in charge of routing, bridging and network functions and do not have power limitations. On the other hand, mesh nodes can be stationary or mobile and require the efficient use of their energy supply like MANET nodes. WSNs are made up of a set of wireless sensor nodes that are usually deployed in hostile environments and are used for event detection (e.g., temperature and pressure measure, etc.). These sensors can perform some processing over the information obtained and transmit the data through the network, which provides the final user with a better understanding of the current state of the environment.

Compared to common wireless mobile devices, WSN nodes are smaller, less expensive, and have fewer hardware characteristics and power consumption. However, due to their operating nature, once a node has depleted or damaged its battery, it may never be retrieved.

2.2.2. Fog Computing

It is possible to deploy an architecture with heterogeneous devices and to extend the characteristics of the cloud to edge devices, avoiding the network bottlenecks. Since the edge devices consume data that can provide elastic computing, they can be connected ubiquitously and share their resources, in many cases collaboratively, achieving the primary purpose of fog computing [15].

Fog computing is the relationship between edge devices and network core (cloud) [16–18]. Fog computing resources are the same for all nodes (networking, computing, and storage), and in most cases, share the same logical abstractions of virtualization and multi-tenancy. Fog computing is typically used for low latency and geo-distributed applications (e.g., sensor networks and surveillance systems). Additionally, they are used in large-scale distributed control systems as smart grids, smart buildings and smart farming.

Inside the fog environment, tenants perceive the resources as dedicated. This is the result of sharing resources, using virtualized file systems and a network infrastructure (e.g., Software Defined Networks (SDN)) [19]. In Reference [20], the author proposes a set of features to deploy services and an application within a fog environment, including node configuration, nodal collaboration, resource/service provisioning metrics, service level objectives, applicable networking system, and security.

The fog computing is possible to implement and design with the addition of MANET features as the duality node/router on each node, without infrastructure to exploit the computing resources on the fog environment and scale the scope to cloud computing. In References [21,22], some architecture is proposed to deploy services and share resources. The main objective is to optimize the Quality of Service (QoS) of heterogeneous resource attributes and applications known as a fog colony similar to an ad hoc cluster. In Reference [23], the infrastructure as a service approach is introduced for fog and cloud computing. The authors proposed a resource pool manager to detect and resolve deadlock and manage resources within the fog environment, and to achieve this objective, they employed a so-called free space fog resources. In this case, our model is oriented to probe the feasibility to use a fog computing or Ad Hoc Network as a cluster of resources on the paradigm of the mobile cloud.

In Reference [24], another model was proposed that employs three tiers: the first, Things tier that manages wireless sensors, actuator networks, and mobile devices. These devices send information to the Fog (second tier), which includes the fog nodes (switches and routers) that link the system to the Cloud (third tier). This approach implies more computation for complex applications that cannot be executed by a fog node alone. The authors present an algorithm called unit-slot optimization, based on Lyapunov's optimization technique to balance the average response time, average cost, and average number of application loss (three-way trade-off).

These prototype sets are based on cloud computing and edge network devices to reduce latency and improve the resources time consumption. The control is reduced and system changes are limited; however, they can be managed by each cloud's control systems and edge devices. Our approach is a prototype based on this kind of architecture in order to validate the feasibility of management and to create virtual operative systems as Wireless Distributed Computing.

In some cases, the most interesting element in a fog computing architecture is the orchestrator [25]. The orchestrator is a useful abstraction to manage and control distributed systems. It observes, decides, and acts, through policies defined by the system administrator, an artificial agent, or some script. This kind of computing artifact, used at different tiers of computing infrastructure and services [25], is commonly shown in systems such as Software Defined Networks (SDN), Network Virtualization Functions (NFV), and Cloud computing application resource management. These three technologies converge in the same artifact: the orchestrator, as shown in Figure 1.

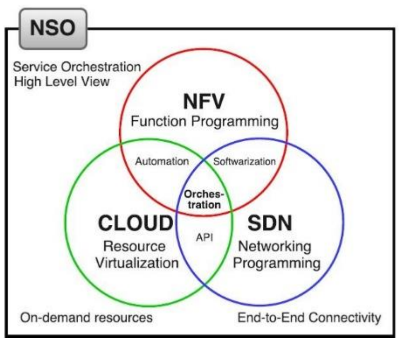


Figure 1. Orchestrator technologies [26].

The development of this kind of computational artifact adds a new set of policies to manage fog nodes and interact with cloud nodes. The framework for the management and orchestration of smart cities applications is presented, following the guidelines of the European Telecommunications Standards Institute (ETSI) and NFV Management and Orchestration (MANO) architecture. The fog orchestrator is responsible for managing the life-cycle of micro-services, monitoring and analyzing the system, generating an interface with the fog decision model, and registry of the new VNF and Network Services (NS).

Other architecture used to deploy distributed wireless systems such as the Wireless Sensors Networks (WSN) as a low cost computing system that provide scalable features at communication, architecture and resources, for the frameworks to manage the sensors and the communications, the theoretical solutions are based on algorithm that evaluate the performance and state the sensors, while they perform measurements simultaneously, the comparison of distributed algorithms is presented at Reference [27] with a simulation tool on two different network topologies.

2.2.3. Operating Systems for Distributed Systems

For deploy distributed systems such as IoT/Fog environments, it is common to use embedded system such as main devices, exist software and operating systems to manage the computing resources and communications on these devices. Some of them are for general purpose end devices used for deploy services and to create environments between end devices and services on the cloud or local servers. A taxonomy for IoT operating systems is proposed at Reference [28], based on the amount of computing resources, communication features, energy consumption, multi threading capability and support programming languages.

An example of an Iot operating system is RIOT ([29]), which is based on microkernel architecture to manage computing resources on IoT devices creating resource abstractions on each board. The operating system mapping the CPU and creating a software abstraction based on APIs for the create modular application, which is a compact and modular model to improve the requirements from IoT environments. However, there is no distributed approach to share and manage resources on an IoT environment or network cluster, instead of allowing the optimal use of board resources and exploiting the device capabilities.

Other approaches for IoT architecture include the frameworks that help manage some resources or to control the processing in some of the system's member nodes. A case study is the prototype of a service-oriented middleware CoTWare [30] for the management of large-scale IoT applications, which visualize resources such as services to deploy applications, and this prototype was made with Arduino devices and three computers to simulate service calls under the RESTful APIs model.

Another framework based on layer architecture for Iot proposes the use of Named Data Networking for edge architectures and cloud computing [31]. In the NDN architecture, the content is treated as the first-class citizen rather than host and names, which are used for network layer communication instead IP addresses. This paper shows, in order to solve the challenges of real-time services and to improve the performance of real time services, a combination of well-known innovative technologies Named Data Network and Edge Cloud Computing to reduce the latency between services and to resolve the user request.

The new approaches are based on container technologies such as Docker to deploy distributed services that are joined with a middle-ward, as shown in Reference [32], which adds concepts such as real-time containers, presented in Reference [33]. Finally, the most similar approach to this paper was simulated with the GNS3 Simulator on a Fog Computing architecture based on virtual objects, created on the simulator as computing resources, sensors and applications. The model and results are included in Reference [34].

3. S.O.V.O.R.A. Wireless Distributed Systems

One of the main problems within the architectures and structures presented so far is data management, i.e., how can you control and improve the operation of a system that has a distributed behavior? It is relevant to highlight that a distributed system operates at different stations. This means that the resources are in different machines and users are not aware of the interactions between them [35]. These distributed characteristics bring a variety of challenges for information management and monitoring the entire status of the system. It is key to identify which services require information consistency, availability, and partitions tolerance [36].

3.1. Architecture

The architecture used on the prototype is one of the microservices. The software architecture provides autonomy, isolation, and the definition of a task for each service or software module, so the set of microservices composes the overall software architecture for the Distributed Wireless Operating System. The microservices architecture is based on a simple concept: the creation of a system from the set of services, each one of them with its small, independent, isolated, scalable, and resilient to failure data. The decomposition of the systems into a discrete and isolated subsystem is reported in well-defined protocols.

The key factor in microservices is the isolation, which is a prerequisite for resilience and elasticity. It requires asynchronous communication to decouple the limits of microservices in time to allow for concurrency and distribution. In addition, it requires Space that allows the services to move around the system (mobility).

Figure 2 shows a set of microservices defined for the local and the OR agents to create the minimum components to deploy S.O.V.O.R.A. on a MANET: the client and multi-thread servers for wireless communication and message passing, the module to monitor resources, a monitor device, log information and command execution of some message from the orchestrator or the user.

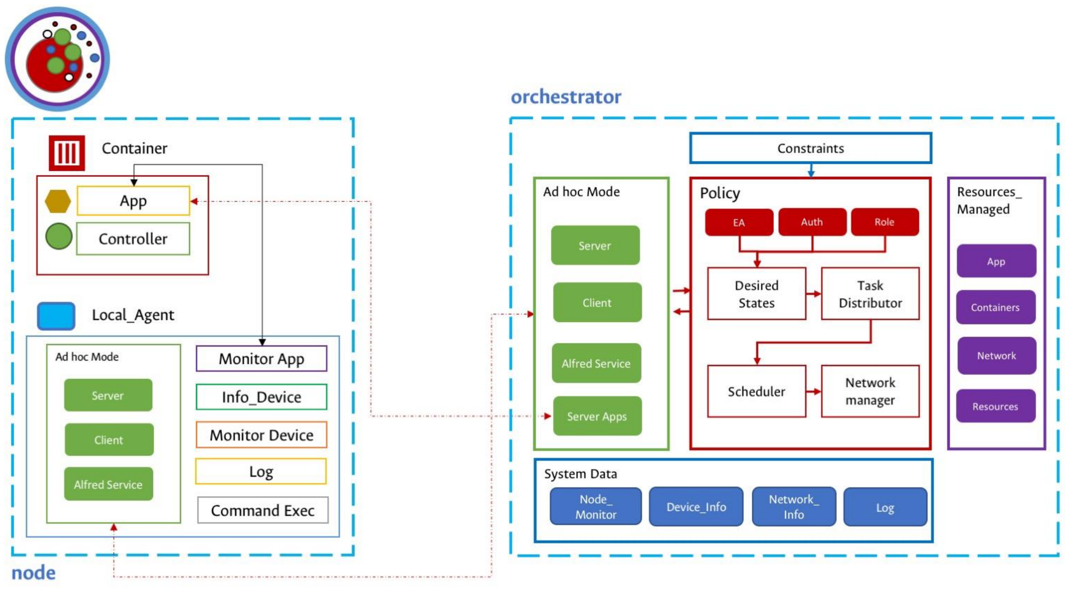


Figure 2. Local Agent-Orchestrator Model.

The primitives of S.O.V.O.R.A. are oriented to the distribution of resources over a MANET cluster to define the need for communication schemes. The distribution of tasks in the system makes it necessary to identify the elements called objects:

- Nodo (Node): Represents the physical devices.
- GGlobal (Ggroup): Represents a gorup of physicial nodes on the same MANET.

- Group (Group): Represents a set of local agents.
- Local Agent (Agl): Represents an instance of local agent on the physical node. On each node can exist one or more instances of LA.
- Orchesrator (Orch): Represents a global controller the system. On each MANET cluster can exist one or more instances.

Table 1 shows the commands that are available for users on the wireless operating system.

Table 1. S.O.V.O.R.A. terminal commands.

Local Agent	
is_alive()[options-node, ip]	Verify the nodes on line and linked with SOVORA
node()[options-node,ip]	show information about the node resources, IP and UUID
process()[options-node,ip]	Show the process running on execution time on local node
Orchestrator	
sstatus [options][ip, port]	Shows the system status
nstatus [options][ip, port]	Show the MANET status
SOVORA	
server [options][ip, port]	Create a multi thread server on specific node
global()[options-ssid]	Shows the node member on the MANET cluster
org()[options-ssid]	Shows the network organization
gglobal()[options-nodes]	Create a communication group between local agents
disk()[options-app]	Provision disk space
cpu()[options-app]	Provision a thread
mem()[options-app]	Provision distributed memory
io()[options-app]	Provision I/O devices connection
pidg()[options-nodes]	Shows the process on all systems nodes
dprocess()[options-node,ip]	Create a process on specific node
mess()[options]	Create a broadcast message
clock()[options]	Set the systems clock

3.2. *The Orchestrator*

The OR is a process that runs periodically on the fog controller node(s) in the user space at the top of the OS. As a centralized controller, the main tasks are: (i) to collect the status information from each node to build an overview of the distributed system's status, and (ii) to send the commands to compose the workload or user requests to the distributed applications.

The Ad hoc mode supports communications between the local agents and the applications on the system. It has the following microservices:

- Server: It is a microservice that create a server multi-thread for the communication between nodes and applications.
- Client: A microservice managed by the local agent that creates a multi-thread client to collect information about the node applications, send messages to deploy applications, and set the throughput and the resources consumption in each node.
- B.A.T.M.A.N. protocol: Proactive network protocol that works on ad hoc mode on IEEE 802.11 standard. The Protocol allows for communication between nodes and orchestrator due to the best route based on transmission quality factor. The communication is a multi hop on the cluster.

- Alfred Service: A microservice used as module to enable the node discovery service, on the network through B.A.T.M.A.N. protocol that allows signaling and to identify the orchestrator on the network.
- Server Apps: It is a microservice that receives the information or results from the applications and sends it to the user or destination node.

The orchestrator has the policies with the information about the current state of the system, which is the desired state. To reach it, the task distributor microservice program the tasks via the scheduler and the network manager that has the information about the network state and the identification of the nodes.

The resource manager is a module that has microservices to monitor and make the assignment of computing resources in the network, deploying more services among MANET clusters. Microservices are applications, containers (Docker containers used to deploy distributed applications), networks, and computing resources.

3.3. The Local Agent

Similarly to the OR, the LA is a process that runs periodically in the user space at the top of the OS as one or multiple instances on the node to manage different resources or applications. The LA interacts with the OS to collect information related to the status of hardware resources, running applications, and MANET. Perform resources and workload control are based on that information. As shown in Figure 3, the LA has a set of microservices to allow for interaction with the running applications and to carry out monitoring and control at the application-level, estimating the computing resources consumption of the node, communicating periodically with the OR, sharing the status of the node and receiving execution requests. The modules are as follows:

- Ad hoc Mode: This module enables the communication between the LA and the OR to send and receive the commands or the log information with the Alfred service.
- Monitor App: A microservice that monitors the application performance. A controller links the LA to applications designed for use in the system.
- Information: A microservice that collects the local information such as computing resources (CPU, Memory, Storage, I/O) and applications state, and the LA sends it to the OR.
- Monitor Device: A microservice that shows the information in real-time executed as a thread.
- Log: A microservice that stores all logging information about resources, messages, network interface state, and local and global interactions in each node or instance.
- Command Exec: A microservice that receives the messages from the users or the OR to execute, stop, or exit some distributed application.

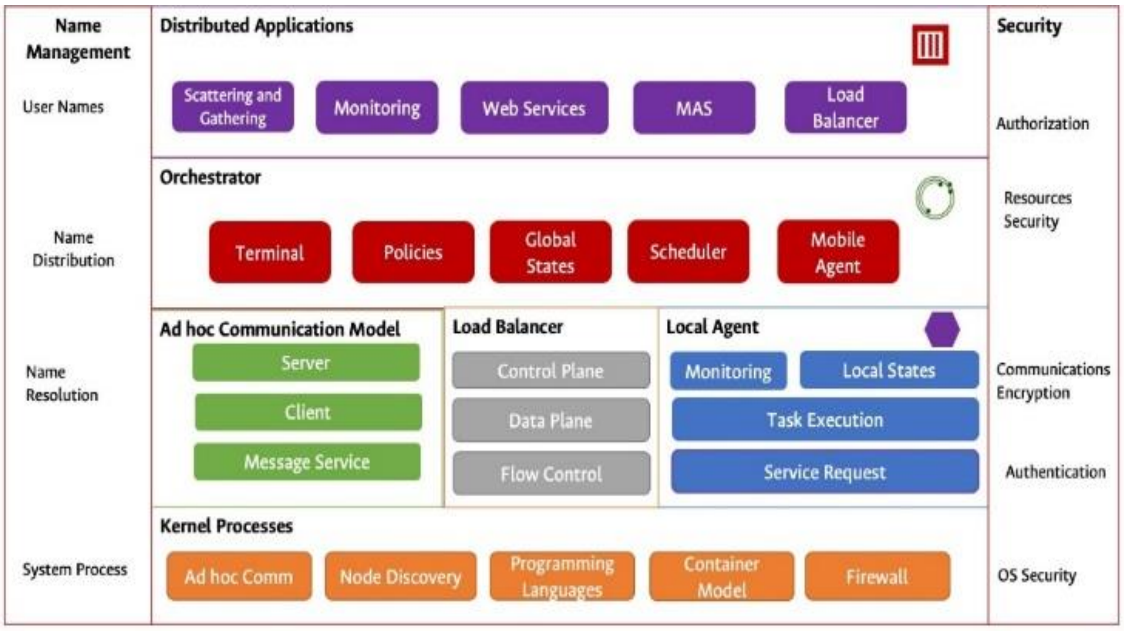


Figure 3. S.O.V.O.R.A. proposed Model.

Algorithm 1 shows the LA routine and the tasks carried out to keep the system in operation and communication between member nodes of the system and the OR. The input for the algorithm are the main services for the framework as the communication in the Ad hoc mode, as well as the OR discovery service and the communication servers with the different entities signalized in the system. Subsequently, the applications are deployed in docker containers, where the application controller is located and deployed, considering the number of cores available for execution.

Algorithm 1. Local Agent Algorithm.

```
   Data: Communication node Status
   Input: Communication Node Status
   Input: Ad hoc Mode (B.A.T.M.A.N.) enable
   Input: RUN ADHOC MODE
   Input: RUN DISCOVERY SERVICE
   Input: Run Log file
   Input: Run ThreadServer(STATES)
   Input: Run ThreadServer(EXEC)
   Input: Run ThreadServer(REGISTRY)
   Input: RUN ThreadDocker APPS
 1 if serverExecThread==true then
 2     params:Orch(Poldata) ;
 3     ExecPol (params) ;
 4 end
 5 if ServerNodeInfo ServerExecApps == true then
 6     for k in cores do
 7         create_container ;
 8     end
 9 end
10 exit
11 while N = 0 do
12     int i=0;
13     write node_info;
14     if i == 0 then
15         for m in cores do
16             cpu_acum{m} ;
17             container_acum{m} ;
18         end
19     else
20         for k in containers do
21             cpu_acum ← cpu_acum_current;
22             cont_acum ← cont_acum_current;
23             Get Status(%CPU,Thr_App, Thr_Cont, Energy, Mem, Disk);
24             info_node ← (Get Status,cpu_acum, cont_acum);
25         end
26     end
27     save LocalLog;
28     i++ ;
29     if Orchestrator enable == true then
30         send orchestrator (node_info);
31     end
32 end
```

The main process occurs on lines 5 to 28, where arrangements to save the history of resources are consumed in the node. For example, the %CPU is the resource to be saved as a value. This task is done globally at the node level and locally at the container level. In the same way, the performance of the running applications is monitored. These values are stored and sent to the OR (lines 29 to 31). In this process, it is important to highlight the execution of policies (lines 1–4) where there is a server thread that waits for the instructions to execute the application policies contained in the LA.

3.4. Network

In the proposed empirical model, the MANET was deployed with the B.A.T.M.A.N. protocol [37] that works on an ad hoc mode in the IEEE 802.11 standard set on the wireless NIC. The self-configuration is based on the avahi-daemon for setting an IP address dynamically. The discovered node uses the ALFRED messaging service [38] that publishes in the MANET and the information of each node as messages with an ID. Only nodes in the same cluster or ESSID can reach the messages on the network.

While each node has a label to link and register with the OR, the LA enables the ALFRED protocol to send information over the MANET (Figure 4). The messages that pass between the nodes are through MANET control messages, while the ALFRED protocol messages are used as a json file with information about the node label, local states, available resources, application states, and network status. The information is collected by the local agent and sent to the OR to be analyzed and saved in its repository. The information collected is used to manage resources, applications, and tasks in the distributed wireless OS. The B.A.T.M.A.N. routing protocol is useful to reduce the latency and to know the location of the nodes according to the number of jumps between the nodes and the MAC address of the wireless NIC.

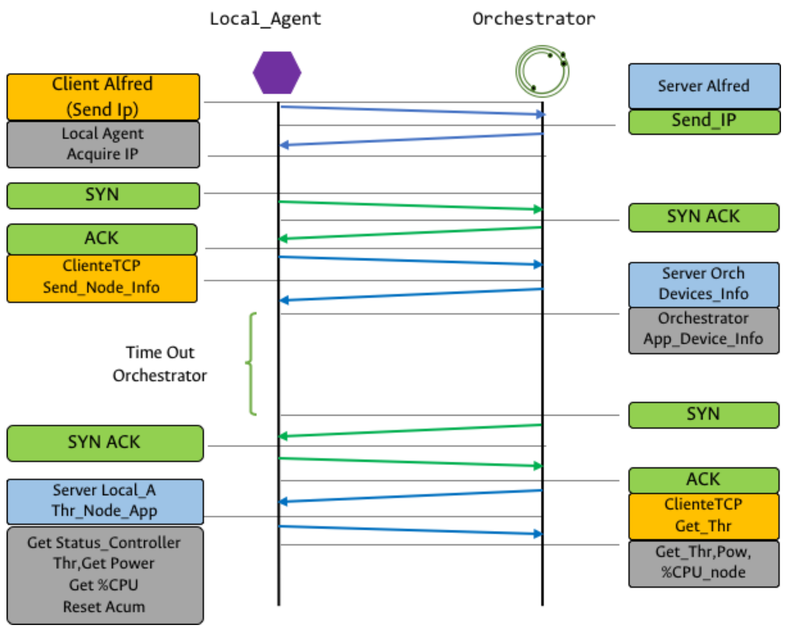


Figure 4. Local Agent-Orchestrator Communication.

The second phase is controller-LA communication, as shown in Figure 5. This communication is made from the LA to the distributed application; in this case, the application is located in a Docker container with the controller, which links the applications to monitor the performance. Thanks to this communication, it is also possible to adjust the operation speed of the applications, and in this sense, the number of resources used by the application. This scheme allows for monitoring at three levels: the hardware, the state of the mapped resources, and the applications deployed in the system.

The model proposed by S.O.V.O.R.A. is modular, despite being deployed on top of the operating system, and there is a hierarchy of layers (Figure 3). The first modules are necessary for the implementation of the wireless ad hoc communication mode with the B.A.T.M.A.N. routing protocol. The node discovery module works on the ad hoc mode using the messaging service provided by ALFRED. For deploy applications, the Community Edition (CE) docker engine, and the micro-services architecture are used to create, deploy, and monitor applications. The modular model allows for the transition from edge networks to cloud computing.

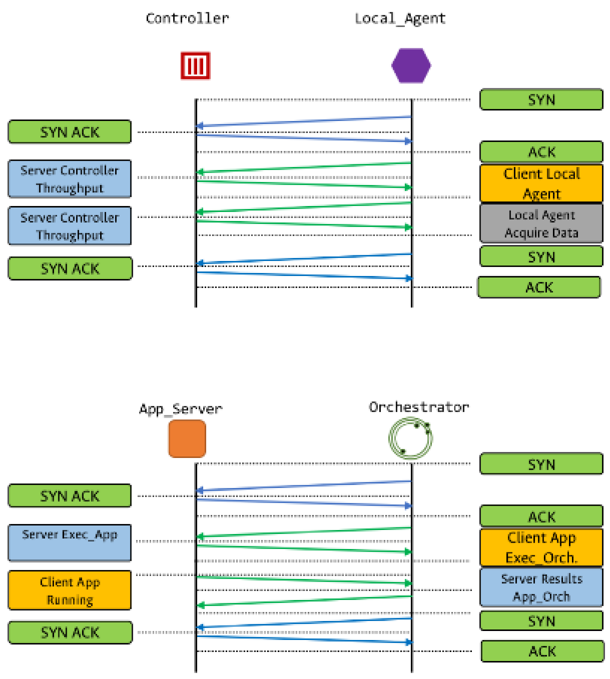


Figure 5. Local Agent Applications Communication.

The second layer is composed of three elements: the communication module that connects to the ad hoc kernel module and generates multi-thread servers and client services in the MANET; the load balancer that defines the abstractions of the control and data planes through the flow control strategies; and the local agent that manages the other two elements of the layer are present in each node (minimum one instance per node), and this also performs the execution and monitoring of tasks and service requests from users.

Monitoring local resources and application states is one of the basic services, and the OR executes this feature. The working layer is deployed in a single physical node for the control of the operating system (it is possible to deploy one or more ORs for MANET cluster). The last layer creates a simple distributed surveillance application with a monitoring service inside with some micro-services such as monitoring and web services deployed in docker containers.

The prototype contains two artifacts for its operation, the LA agent and the OR agent, both working over a MANET. Considering that the nodes can work as routers and clients, the artifacts use the Better Approach To Mobile Adhoc Networking (B.A.T.M.A.N.) as routing protocol, giving this model the ability to mitigate problems associated with coverage and hidden nodes, as well as allowing node discovering.

The LA acts with two roles: first, as a reactive agent, which executes the resources control policy, this model allows for controlling the applications to be deployed. In this case, each application is in a container and has a controller for its deployment. The model allows for having one or more local agents in one device to control resources and to monitor the applications. Second, as a deciding agent, the agent decides to apply the policies by sending them across the MANET. The local agent decides the amount of resources in the application container for the applications. The second element is the OR is an agent that can be deployed on unique node or each node on the network. It decides if an application is executed and continues the load balancing policy for the devices on the network. The agent has information about the global and local states of each device (CPU throughput, memory consumption, available bandwidth, network availability, and apps performance). This information is periodically received from local agents that inform the status of each node. In this environment, the OR model offers the possibility of monitoring different nodes and applications, allowing for the control of variables such as energy consumption, application distribution, and computing resources. These experiments only use one instance of an orchestrator agent due the limited physical nodes to manage the network (the experiment has six physical nodes).

The OR also allows for the control of distributed wireless operative systems and their interactions, e.g., communication between nodes, allocation, and assignment of computing resources on all cluster, distribution and management of throughput of the applications distributed on each node.

4. Experiment and Results

To validate the report of this work, the abstraction of a distributed wireless operating system was implemented, which dynamically manages computing resources in Devices on a MANET.

The experiment and test bed were based on the embedded devices referenced in Table 2 and the goal was to demonstrate the feasibility and viability and of this approach. These devices have LA and OR similar to the distributed applications in docker containers (launched and monitored by the LA). Global states are sent to a multi-thread server in the OR and are stored for use when the tasks are distributed in the system (workload). The first part of the deployment is the MANET creation and the discovery of nodes and resources. Then, the OR processes the user's requests as a workload for the wireless OS. The test application was designed as a distributed video-conferencing application segmented into containers, as shown in Figure 6. Each container has a microservice of the application processing, as shown in Table 3, which comes from the benchmarking suite MiBench [39]. In addition, each application is launched in a core or a different node depending on the OR policy. The throughput is set in the same way to manage and distribute workloads evenly on the network nodes; the results are sent to the source or a client for display. This model makes it possible to control computer resources such as energy, and to reduce the processing time of distributed applications; the power was measured with the smart power 2 [40] with more granularity and precision to create a consumption power model for each device on the MANET.

Table 2. Board information.

Device	Architecture	Processor Fam.	Chip	Cores	Arm_Freq (MHz)	Gpu_Freq (MHz)	Mem (MB)	Wifi_Chip
RPi 3	ARM	ARM Cortex-53	Broadcom BCM2837	4	1200	400	1024	CypressCYW43438
RPi zero w	ARM	ARM11	Broadcom BCM2835	1	1000	400	512	CypressCYW43438
Rpi 2	ARM	ARM-Cortex-A7	Broadcom BCM2836	4	900	250	1024	External
Odroid	ARM	ARMV8	Samsung Exynos5422	8	2000	533–295	2048	External

Table 3. Selected applications for the use case.

Application	Algorithm	Resources
Security	SHA	CPU, Mem, Bandwidth
Image Processing	JPEG	CPU, Mem, Bandwidth
Audio	ADPCM	CPU, Mem, Bandwidth
Image Processing	SUSAN	CPU, Mem, Bandwidth

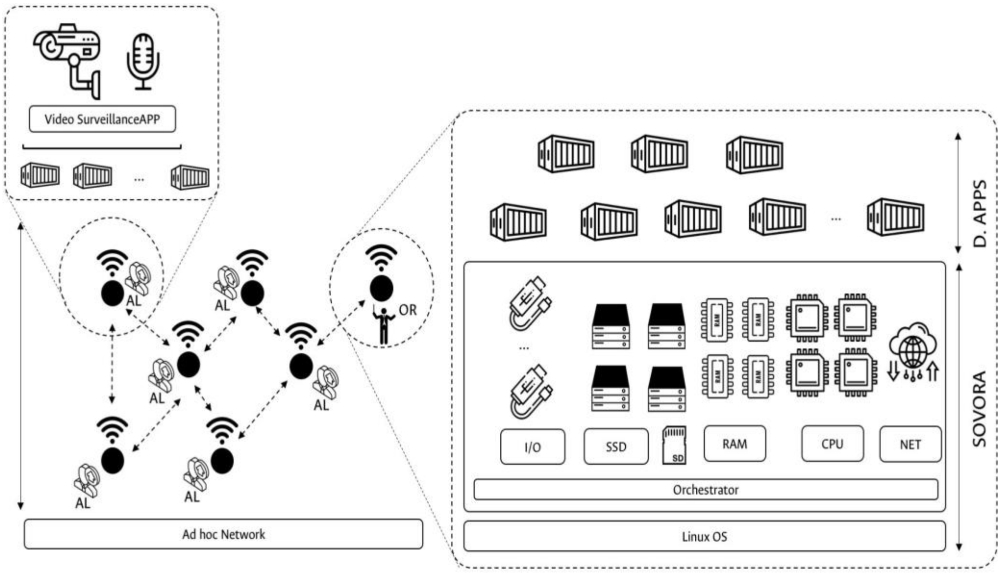


Figure 6. Experiment Model.

4.1. Proposed Testbed

To characterize the energy consumption, an experimental validation was carried out with the MiBench benchmarking suite, with the algorithms listed in Table 3 to measure the energy consumption at different CPU rates, which were controlled through processor frequency. The linearization of the output function can be seen in Figure 7. This generated output function is the basis of the OR policy to select the destination node in terms of the task to be performed on the network or application selected by the user.

In the same way, communication in the ad hoc mode was validated through the proactive routing protocol B.A.T.M.A.N. With the transmission of data (14 MB for test) validating the routes and the TQ value—Transmit Quality—as an additional factor for the selection of the destination node, taking the measurement of power consumed by each transmission carried out; the results can be seen in Table 4 for network protocol and in Table 5 for energy on a RPI 3.

The OR monitors the network nodes' status and its resources, creating a virtual abstraction of the resources mapping on each node and the amount of resources free for use on the distributed wireless OS and managing it. The test policy is quite simple but it can be customized, for the purposes of the experiment, when mapping the available resources from the information provided by each local agent, and it is stored in the orchestrator which compares it with the information referring to the energy consumption model, the TQ value and the state of charge of the processors of each node. Based on this information, it creates a workload and dispatches it depending on the state of resources and the available route for delivery of the task.

Table 4. Test Ad hoc-batctl/batman-adv RPi 3.

RPi_3batctl	14 MB Rpi_Zero	14 MB Odroid
%CPU	100.00	100.00
Time (s)	10.19	10.54
Power (mW)	2211.88	2230
Energy (J)	2211.88	2230
Throughput (MB/s)	0.8893	1.7
MB Send	7.605	18.5

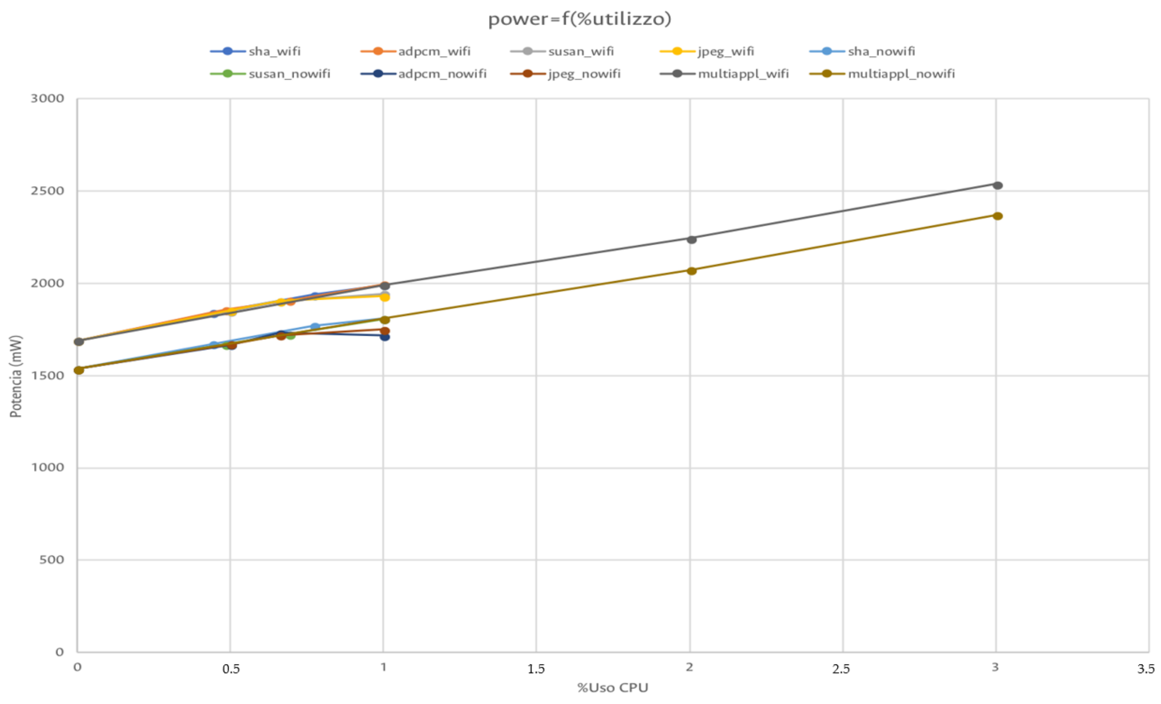


Figure 7. Experiment Model.

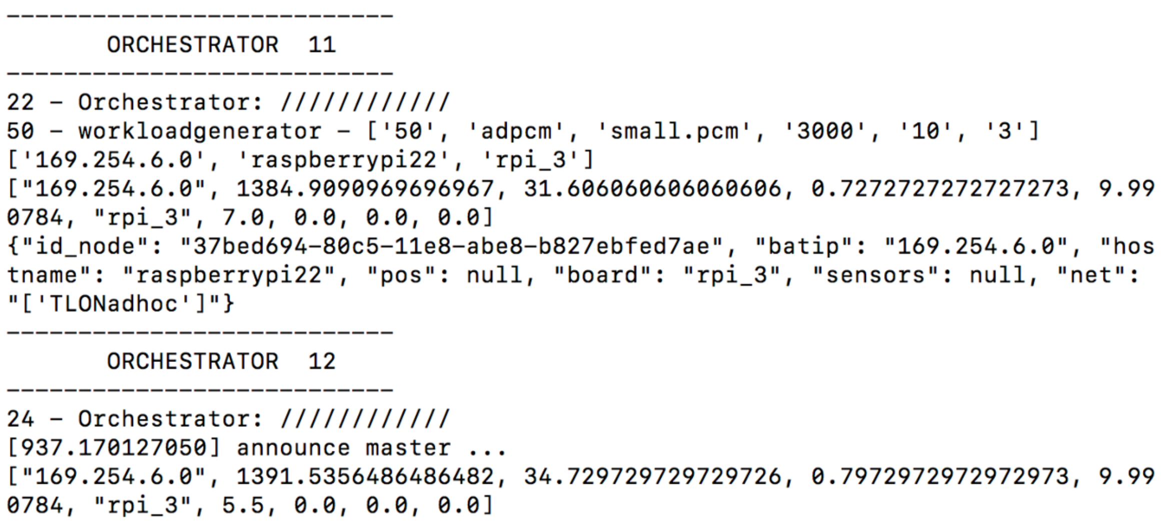
Table 5. Power Consumption RPI 3 + Wifi.

POWER + WIFI RPi 3				
APP	**IDLE**	**50%THR_MIN**	**75%THR_MAX**	**THR_MAX**
SHA	1689.95	1839.75	1938.41	1991.14
ADPCM	1689.95	1856.52	1906.49	1995.51
SUSAN	1689.95	1850.2	1902.81	1942.54
JPEG	1689.95	1852.39	1905.52	1929.54

4.2. Logs and Events

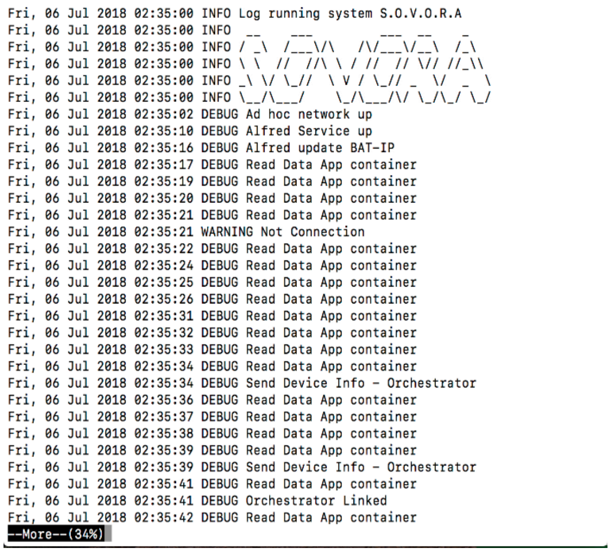
The OR main goal is to manage, with equality, the resources available in the MANET cluster depending on the user's requests and the global status of the system. For the users, a simple terminal as an interface to send the requests to the system to processing with the OR is available. Figure 8 shows the output of information collected for the OR of the network, OS resources available and tasks running on each node.

Based on the events and the information generated by all nodes sent through the local agent (Figure 9), the operating system abstraction generates an event log (Figure 10) with data of connectivity state, application status and workload of each of the local agents in the system. It is important to indicate that, given the number of devices, the experiment can be performed with a single orchestrator and the local agent instances in each of the member nodes of the ad hoc network. This information is local and sent to the orchestrator who is signaled through the network as the only member with the correct information of the system, making consensus between the members of the network and the operating system for its operation. If there are more orchestrators, the raft consensus algorithm [41] is used to guarantee the integrity and availability of the information among the local agents that are members of the network.



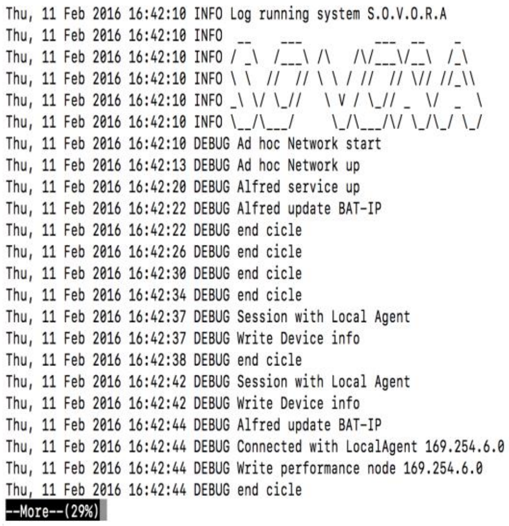
```
--------------------------
       ORCHESTRATOR  11
--------------------------
22 - Orchestrator: ////////////
50 - workloadgenerator - ['50', 'adpcm', 'small.pcm', '3000', '10', '3']
['169.254.6.0', 'raspberrypi22', 'rpi_3']
["169.254.6.0", 1384.9090969696967, 31.606060606060606, 0.7272727272727273, 9.99
0784, "rpi_3", 7.0, 0.0, 0.0, 0.0]
{"id_node": "37bed694-80c5-11e8-abe8-b827ebfed7ae", "batip": "169.254.6.0", "hos
tname": "raspberrypi22", "pos": null, "board": "rpi_3", "sensors": null, "net":
"['TLONadhoc']"}
--------------------------
       ORCHESTRATOR  12
--------------------------
24 - Orchestrator: ////////////
[937.170127050] announce master ...
["169.254.6.0", 1391.5356486486482, 34.729729729729726, 0.7972972972972973, 9.99
0784, "rpi_3", 5.5, 0.0, 0.0, 0.0]
```

Figure 8. Orchestrator Info.



```
Fri, 06 Jul 2018 02:35:00 INFO Log running system S.O.V.O.R.A
Fri, 06 Jul 2018 02:35:00 INFO  __    ___         ___  __   _
Fri, 06 Jul 2018 02:35:00 INFO / _\  /___\/\   /\/___\/__\  /_\
Fri, 06 Jul 2018 02:35:00 INFO \ \  //  //\ \ / //  // \// //_\\
Fri, 06 Jul 2018 02:35:00 INFO _\ \/ \_//  \ V / \_// _  \/  _  \
Fri, 06 Jul 2018 02:35:00 INFO \__/\___/    \_/\___/\/ \_/\_/ \_/
Fri, 06 Jul 2018 02:35:02 DEBUG Ad hoc network up
Fri, 06 Jul 2018 02:35:10 DEBUG Alfred Service up
Fri, 06 Jul 2018 02:35:16 DEBUG Alfred update BAT-IP
Fri, 06 Jul 2018 02:35:17 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:19 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:20 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:21 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:21 WARNING Not Connection
Fri, 06 Jul 2018 02:35:22 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:24 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:25 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:26 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:31 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:32 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:33 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:34 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:34 DEBUG Send Device Info - Orchestrator
Fri, 06 Jul 2018 02:35:36 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:37 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:38 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:39 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:39 DEBUG Send Device Info - Orchestrator
Fri, 06 Jul 2018 02:35:41 DEBUG Read Data App container
Fri, 06 Jul 2018 02:35:41 DEBUG Orchestrator Linked
Fri, 06 Jul 2018 02:35:42 DEBUG Read Data App container
--More--(34%)
```

Figure 9. Local Agent Log.



```
Thu, 11 Feb 2016 16:42:10 INFO Log running system S.O.V.O.R.A
Thu, 11 Feb 2016 16:42:10 INFO  __    ___         ___  __   _
Thu, 11 Feb 2016 16:42:10 INFO / _\  /___\ /\   /\/___\/__\  /_\
Thu, 11 Feb 2016 16:42:10 INFO \ \  //  // \ \ / //  // \// //_\\
Thu, 11 Feb 2016 16:42:10 INFO _\ \/ \_//  \ V / \_// _  \/  _  \
Thu, 11 Feb 2016 16:42:10 INFO \__/\___/    \_/\___/\/ \_/\_/ \_/
Thu, 11 Feb 2016 16:42:10 DEBUG Ad hoc Network start
Thu, 11 Feb 2016 16:42:13 DEBUG Ad hoc Network up
Thu, 11 Feb 2016 16:42:20 DEBUG Alfred service up
Thu, 11 Feb 2016 16:42:22 DEBUG Alfred update BAT-IP
Thu, 11 Feb 2016 16:42:22 DEBUG end cicle
Thu, 11 Feb 2016 16:42:26 DEBUG end cicle
Thu, 11 Feb 2016 16:42:30 DEBUG end cicle
Thu, 11 Feb 2016 16:42:34 DEBUG end cicle
Thu, 11 Feb 2016 16:42:37 DEBUG Session with Local Agent
Thu, 11 Feb 2016 16:42:37 DEBUG Write Device info
Thu, 11 Feb 2016 16:42:38 DEBUG end cicle
Thu, 11 Feb 2016 16:42:42 DEBUG Session with Local Agent
Thu, 11 Feb 2016 16:42:42 DEBUG Write Device info
Thu, 11 Feb 2016 16:42:44 DEBUG Alfred update BAT-IP
Thu, 11 Feb 2016 16:42:44 DEBUG Connected with LocalAgent 169.254.6.0
Thu, 11 Feb 2016 16:42:44 DEBUG Write performance node 169.254.6.0
Thu, 11 Feb 2016 16:42:44 DEBUG end cicle
--More--(29%)
```

Figure 10. Orchestrator Log.

4.3. Experiment Results

With the test bed on execution, we ran some application on the system as an operating system task, and then take the measure of energy and application throughput on each node, as shown in Figure 11, which shows the measure of the energy consumption of the Odroid node carried out when using the smart power meter 2. In this case, the peak shows the maximum CPU consumption of these nodes with a simple task and as these measure, which is stabilized through the orchestrator messages and parameters sent to a local agent on this node. In this way, the CPU heartbeats resource, available in the MANET, is managed, and their relationship with the energy consumption is controlled. The peaks occur at the beginning of the execution of each application since the application is always trying to use 100% of the resources. The OR sets the processing rate and sends the desired throughput to the LA that processes it on the docker container.

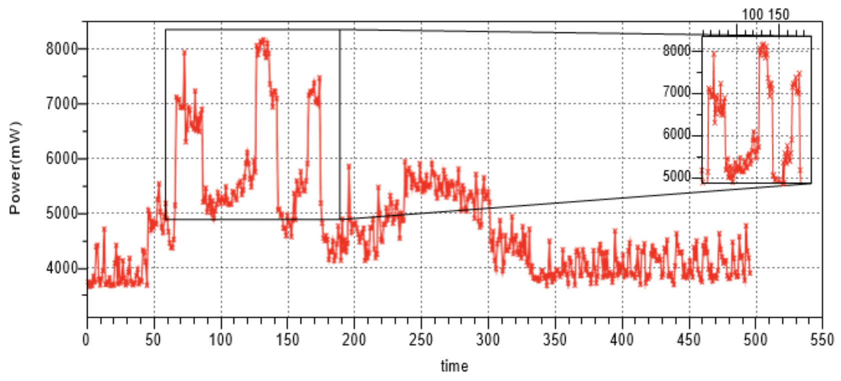


Figure 11. Odroid power consumption.

The first study case is with one node performing the deployment of six containers in each core of the embedded system Odroid (Figure 12). This parallel application can be executed in the Odroid due to its capabilities (Table 2), which have four little architecture cores, four of big architecture, and a variable frequency on each core. This test shows the possible execution of concurrent applications, in this case the set of algorithms from MiBench as workload created by a users that do not have a huge load for processing. These set of applications running concurrently are the model of a distributed application, as a paradigm to deploy services and applications on the wireless distributed operating systems, the traditional application models do not allow for the deployment of services in a distributed way.

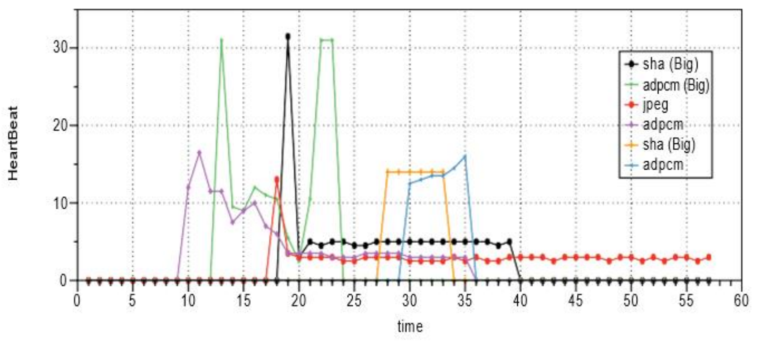


Figure 12. Application performance over Odroid node 0.

The second experiment was performed using five nodes (4 RaspBerry PI-3 and 1 RaspBerry Pi-Zero W) running different applications distributed on them. Nodes 2 and 4 run the SHA encryption algorithm, while nodes 1 and 5 perform images processing using JPEG coding-encoding and node 3 performs ADPCM audio coding. After processing of all this information, it is sent through the MANET and to the LA, which delivers the result to the final user.

Figure 13 shows the results of the execution of each core on each node begin with the maximum throughput and then converge to the throughput indicated for the OR, for instance node 2 begins with a throughput of 16 heartbeats by second and converge at 10 heartbeat at second, at the same way the other nodes reach the performance given for the orchestrator policy.

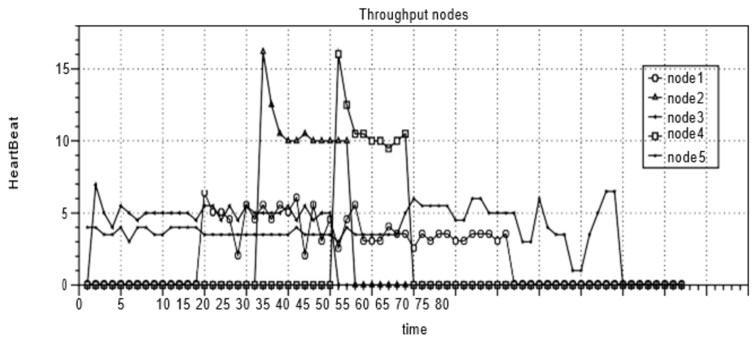


Figure 13. Application performance over each node.

If a new service is required, information about the status of the resources of each node is sent to the OR, which is the source of the application distribution. Figure 14 shows the constant energy consumption of node 2 in the second experiment when a distributed application (containers) is deployed. To send the task, the Orchestrator sends messages every second to neighboring nodes and local agent instances to validate their status based on the CPU consumption and the node's energy measurement, in the same way it validates the available resources to assign the task, and this, with the orchestrator's clock, is a measure of time in the operative system.

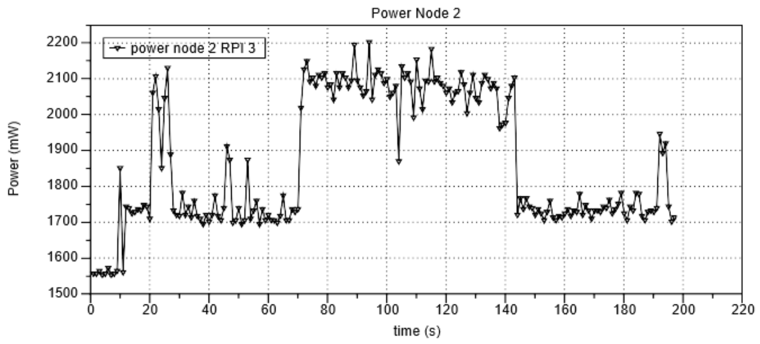


Figure 14. RPI node 2 power consumption.

Figures 15–17 show the cases in which an application was deployed in each core available in each node, and its throughput was controlled to manage energy and CPU consumption. This process allows for the management of the resources distributed in a MANET in a stochastic and dynamic medium. In the same model, it is possible to manage other distributed resources as memory, storage, and I/O devices.

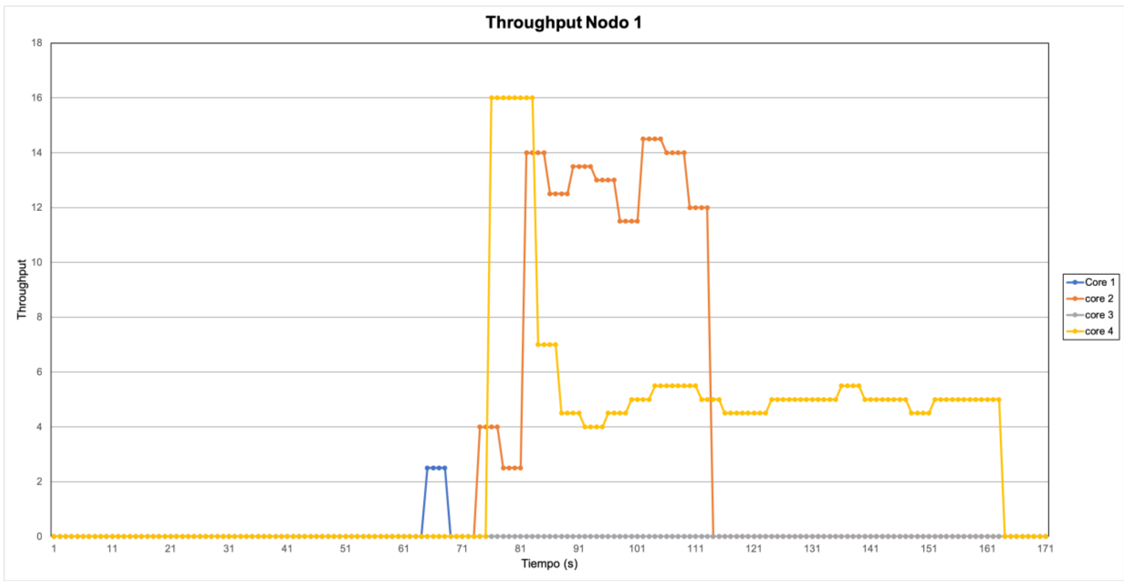


Figure 15. RPI node 1 power consumption.

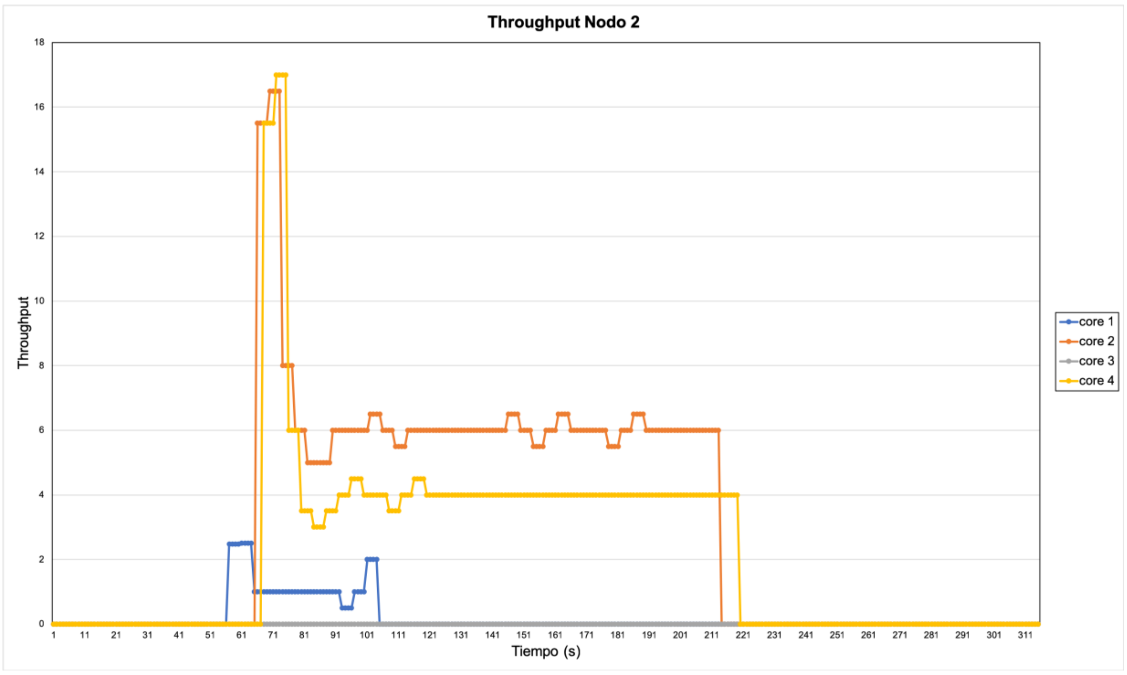


Figure 16. RPI node 2 power consumption.

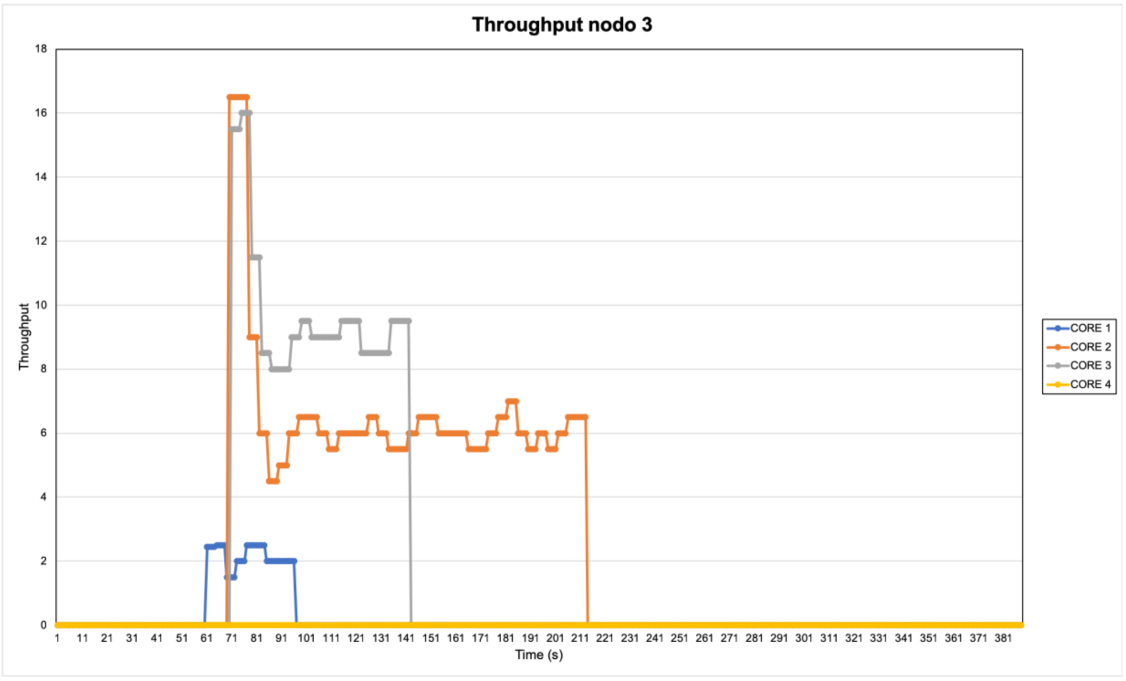


Figure 17. RPI node 3 power consumption.

The OR verifies the global performance and tries to converge the throughput in all the nodes to optimize the resources consumption and the workload distribution among the MANET nodes. Figure 18 shows the performance results of each node in the distributed wireless OS.

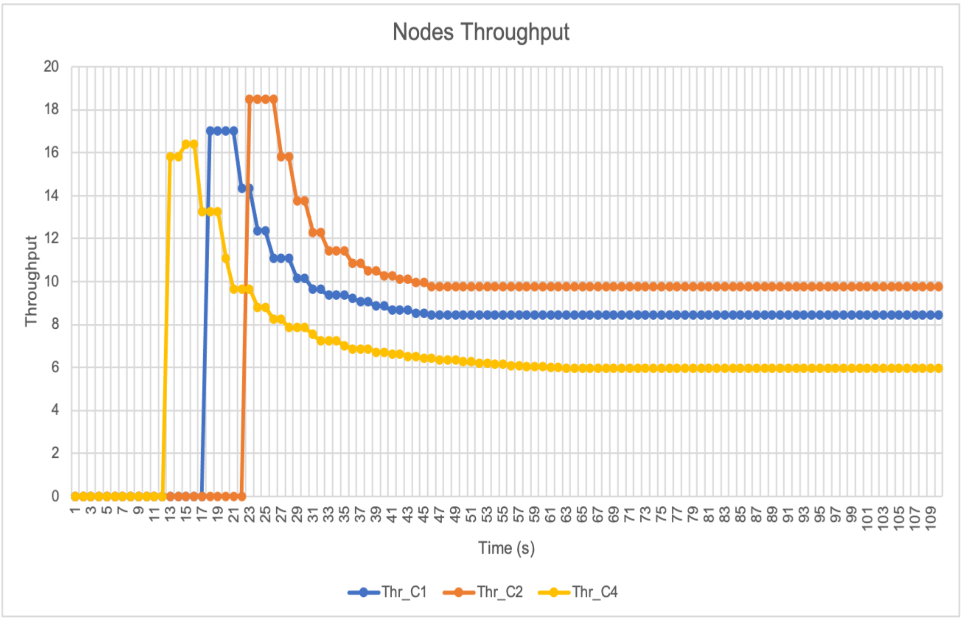


Figure 18. Throughput for node; workload controlled by OR.

With this test scheme, we have demonstrated the possibility of having an implementation of a distributed wireless operating system with two abstractions, the local agent and the orchestrator, which can exist as instances in the nodes of an ad hoc network. In this case, the control of the nodes is by the orchestrator agent based on a policy of energy consumption, which is used to distribute loads through the wireless network, together with the information sent by each node on the state of the resources computation of the nodes and their load at the 468 application level.

This method allows for the generating of an abstraction of the operating system in order to manage distributed computing resources, to create a model to distributed applications, with all of them running on a MANET, even though these models can be replicated on other wireless infrastructures and technologies. Our first approach was with the IEEE802.11x ad hoc mode. Taking advantage of proactive routing protocols as a mechanism for discovering nodes, routes and services available in the S.O.V.O.R.A. without the traditional load in the IEEE 802.11 infrastructure mode.

5. Conclusions

The wireless distributed operating systems are a paradigm that involves the network stochastic and dynamic behavior, due the mobility and the channel shared conditions on wireless networks. On our approach, the nodes on the network are a pool of computing resources that are available for the users in order to deploy applications and services as a distributed operating system. For the deployment, this system is needed to create a model of distributed applications in order to test the complete scenario for a wireless distributed operating system.

Wireless distributed OS have potential in environments such as IoT, fog computing, edge computing, and similar, as they allow for load balancing, sharing and managing computing resources under the mobile cloud model. Our approach uses two kinds of agents: the Local Agent and Orchestrator based on a scheme that can check the local and global data from each node on the network to control the distributed computing resources of the network—in this case, a MANET. Thanks to its auto-configuration properties, this kind of network allows for the deployment of services without infrastructure and to mitigate some routing problems. Due to continuous communication and signaling supplied by the B.A.T.M.A.N. protocol, the proactive routing protocol allows for the distributed application approach, and the deployment of the isolated micro-services in containers on each node as instances, setting the amount of resources needed for the deployment of applications in different nodes of the distributed wireless OS. This is similar to how B.A.T.M.A.N.-adv works on the second network layer, which reduces the traditional network layers workload, if it compares the infrastructure mode on IEEE 802.11.

The model proposed allows for deploy services as the node discovery, node signaling, fail detector, instant messaging and so on, due to the updating of routes on all networks, and the disseminated of routes and system information in all nodes on the network every second, these features allow for the distribution of tasks, to know the status of resources on the network, to create software and hardware abstractions by passing messages onto the agents (LA and OR) on ad hoc mode and to deploy distributed applications.

Author Contributions: Conceptualization, H.Z.C. and J.E.O.T.; methodology, H.Z.C.; software H.Z.C.; validation, J.E.O.T.; investigation, H.Z.C.; resources, H.Z.C. and J.E.O.T.; data curation, H.Z.C.; writing—original draft preparation, J.E.O.T. and H.Z.C.; writing—review and editing, J.E.O.T.; visualization, H.Z.C.; supervision, J.E.O.T.; project administration, H.Z.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received external funding from Systems and Industrial Department , Engineering Faculty, Universidad Nacional de Colombia—Bogotá.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Fitzek, F.H.; Katz, M.D. *Mobile Clouds: Exploiting Distributed Resources in Wireless, Mobile and Social Networks*; John Wiley & Sons: Hoboken, NJ, USA, 2013.
2. Dolui, K.; Datta, S.K. *Comparison of Edge Computing Implementations: Fog Computing, Cloudlet and Mobile Edge Computing*; Global Internet of Things Summit; IEEE: Dublin, Ireland, 2017; pp. 1–6.
3. Ceballos, H.Z. Diseño de un Sub-Sistema de Cómputo Distribuido que permita implementar virtualización inalámbrica para gestionar recursos (Procesamiento, memoria, almacenamiento y dispositivos E/S) distribuidos en una Red Ad Hoc, mediante el modelo de pseudo Estado. Línea de Investigación: Computación Aplicada. Ph.D Thesis, Universidad Nacional de Colombia, Bogotá, Columbia, January 2018.

4. Shahzadi, S.; Iqbal, M.; Dagiuklas, T.; Qayyum, Z.U. Multi-access edge computing: Open issues, challenges and future perspectives. *J. Cloud Comput.* **2017**, *6*, 30. [CrossRef]
5. Bittencourt, L.F.; Diaz-Montes, J.; Buyya, R.; Rana, O.F.; Parashar, M. Mobility-aware application scheduling in fog computing. *IEEE Cloud Comput.* **2017**, *4*, 26–35. [CrossRef]
6. Datla, D.; Chen, X.; Tsou, T.; Raghunandan, S.; Hasan, S.S.; Reed, J.H.; Dietrich, C.B.; Bose, T.; Fette, B.; Kim, J.H. Wireless distributed computing: A survey of research challenges. *IEEE Commun. Mag.* **2012**, *50*, 144–152. [CrossRef]
7. Gao, M.; Addis, B.; Bouet, M.; Secci, S. Optimal Orchestration of Virtual Network Functions. *arXiv* **2017**, arXiv:1706.04762. [CrossRef]
8. Giotis, K.; Kryftis, Y.; Maglaris, V. Policy-Based Orchestration of NFV Services in Software-Defined Networks. In Proceedings of the Conference Network Softwarization, London, UK, 13–17 April 2015; pp. 1–5.
9. Sun, C.; Bi, J.; Zheng, Z.; Hu, H. HYPER: A Hybrid High-Performance Framework for Network Function Virtualization. *IEEE J. Sel. Areas Commun.* **2017**, *35*, 2490–2500. [CrossRef]
10. Santoro, D.; Zozin, D.; Pizzolli, D.; De Pellegrini, F.; Cretti, S. Foggy: A Platform for Workload Orchestration in a Fog Computing Environment. In Proceedings of the Cloud Computing Technology and Science, Sydney, Australia, 11–13 December 2017; pp. 231–234.
11. Asnaghi, A.; Ferroni, M.; Santambrogio, M. DockerCap: A Software-Level Power Capping Orchestrator for Docker Containers. In Proceedings of the Computational Science and Engineering/Embedded and Ubiquitous Computing, Paris, France, 24–26 August 2016; pp. 90–97.
12. Hui, F.; Mohapatra, P. Experimental Characterization of Multi-Hop Communications in Vehicular ad Hoc Network. In Proceedings of the 2nd ACM international workshop on Vehicular ad Hoc Networks, Cologne, Germany, 2 September 2005; pp. 85–86.
13. Akyildiz, I.F.; Wang, X.; Wang, W. Wireless Mesh Networks: A Survey. *Comput. Netw.* **2005**, *4*, 445–487.
14. Akyildiz, I.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. Wireless sensor networks: A survey. *Comput. Netw.* **2002**, *38*, 393–422. [CrossRef]
15. Hu, P.; Dhelim, S.; Ning, H.; Qiu, T. Survey on fog computing: Architecture, key technologies, applications and open issues. *J. Netw. Comput. Appl.* **2017**, *98*, 27–42. [CrossRef]
16. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things. In Proceedings of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16.
17. Bonomi, F.; Milito, R.; Natarajan, P.; Zhu, J. Fog Computing: A Platform for Internet of Things and Analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 169–186.
18. Ning, H.; Li, Y.; Shi, F.; Yang, L.T. Heterogeneous edge computing open platforms and tools for internet of things. *Future Gener. Comput. Syst.* **2020**, *106*, 67–76. [CrossRef]
19. Negash, B.; Rahmani, A.M.; Liljeberg, P.; Jantsch, A. Fog Computing Fundamentals in the Internet-of-Things. In *Fog Computing in the Internet of Things*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 3–13.
20. Mahmud, R.; Kotagiri, R.; Buyya, R. Fog computing: A taxonomy, survey and future directions. In *Internet of Everything*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 103–130.
21. Brogi, A.; Forti, S.; Ibrahim, A. How to Best Deploy YOUR FOG APPLICATIONS, Probably. In Proceedings of the International Conference Fog and Edge Computing, Valencia, Spain, 8–11 May 2017; pp. 105–114.
22. Brogi, A.; Forti, S. QoS-Aware Deployment of IoT Applications Through the Fog. *IEEE Internet Things J.* **2017**, *4*, 1185–1192. [CrossRef]
23. Sood, S.K. SNA based QoS and reliability in fog and cloud framework. *World Wide Web.* **2018**, *6*, 1601–1616. [CrossRef]
24. Nan, Y.; Li, W.; Bao, W.; Delicato, F.C.; Pires, P.F.; Zomaya, A.Y. A dynamic tradeoff data processing framework for delay-sensitive applications in Cloud of Things systems. *J. Parallel Distrib. Comput.* **2018**, *112*, 53–66. [CrossRef]
25. Santos, J.; Wauters, T.; Volckaert, B.; De Turck, F. Fog Computing: Enabling the Management and Orchestration of Smart City Applications in 5G Networks. *Entropy* **2017**, *20*, 4. [CrossRef] [PubMed]
26. De Sousa, N.F.S.; Perez, D.A.L.; Rosa, R.V.; Santos, M.A.; Rothenberg, C.E. Network Service Orchestration: A Survey. *arXiv* **2018**, arXiv:1803.06596v4. [CrossRef]

27. He, S.; Shin, H.S.; Xu, S.; Tsourdos, A. Distributed estimation over a low-cost sensor network: A review of state-of-the-art. *Inf. Fusion* **2020**, *54*, 21–43. [CrossRef]
28. Zikria, Y.B.; Kim, S.W.; Hahm, O.; Afzal, M.K.; Aalsalem, M.Y. Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution. *Sensors* **2019**, *8*, 1793. [CrossRef]
29. Baccelli, E.; Gündogˇan, C.; Hahm, O.; Kietzmann, P.; Lenders, M.S.; Petersen, H.; Schleiser, K.; Schmidt, T.C.; Wählisch, M. RIOT: An open source operating system for low-end embedded devices in the IoT. *IEEE Internet Things J.* **2018**, *5*, 4428–4440. [CrossRef]
30. Al-Jaroodi, J.; Mohamed, N.; Jawhar, I.; Mahmoud, S. CoTWare: A Cloud of Things Middleware. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, USA, 5–8 June 2017; pp. 214–219.
31. Ullah, R.; Rehman, M.A.U.; Kim, B.S. Design and implementation of an open source framework and prototype for named data networking-based edge cloud computing system. *IEEE Access* **2019**, *7*, 57741–57759. [CrossRef]
32. Benomar, Z.; Longo, F.; Merlino, G.; Puliafito, A. Enabling Container-Based Fog Computing with Openstack. In Proceedings of the 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Atlanta, GA, USA, 14–17 July 2019; pp. 1049–1056.
33. Struhár, V.; Behnam, M.; Ashjaei, M.; Papadopoulos, A.V. Real-Time Containers: A Survey. In Proceedings of the 2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Sydney, Australia, 21 April 2020.
34. Li, J.; Jin, J.; Yuan, D.; Zhang, H. Virtual fog: A virtualization enabled fog computing framework for Internet of Things. *IEEE Internet Things J.* **2017**, *5*, 121–131. [CrossRef]
35. Haddad, S.; Kordon, F.; Pautet, L.; Petrucci, L. *Distibuted Systems: Design and Algorithms*; John Wiley & Sons: Hoboken, NJ, USA, 2013.
36. Lynch, N.A. *Distributed Algzorithms*; Elsevier: Amsterdam, The Netherlands, 1996.
37. Neumann, A.; Aichele, C.; Lindner, M.; Wunderlich, S. Better approach to mobile ad-hoc networking (BATMAN). *IETF Draft* **2008**, 1–24. Available online: https://tools.ietf.org/pdf/draft-wunderlich-openmesh-manet-routing-00.pdf (accessed on 8 December 2020).
38. Mesh, P. ALFRED. Available online: https://www.open-mesh.org/projects/alfred/wiki (accessed on 8 December 2020).
39. Guthaus, M.R.; Ringenberg, J.S.; Ernst, D.; Austin, T.M.; Mudge, T.; Brown, R.B. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In Proceedings of the International Workshop on Workload Characterization, Austin, TX, USA, 2 December 2001; pp. 3–14.
40. Odroid, U.K. Smart Power 2. Available online: https://odroid.com/dokuwiki/doku.php?id=en:acc:smartpower2 (accessed on 8 December 2020).
41. Ongaro, D.; Ousterhout, J. In search of an understandable consensus algorithm (extended version). *Retrieved July* **2016**, *20*, 2018.