



UNIVERSIDAD
NACIONAL
DE COLOMBIA

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

Henry Zárate Ceballos

Universidad Nacional de Colombia
Facultad de ingeniería, Departamento de Sistemas e Industrial
Bogotá, Colombia
2018

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

Henry Zárate Ceballos

Tesis o trabajo de grado presentado como requisito parcial para optar al título de:
Doctor en Ingeniería Sistemas y Computación

Director(a):
Ph.D., Jorge Eduardo Ortiz Triviño

Línea de Investigación:
Computación Aplicada

Grupo de investigación en Lenguajes de Programación Distribuidos y Redes de Telecomunicaciones Dinámicas
- TLÖN

Universidad Nacional de Colombia
Facultad de Ingeniería , Departamento de Sistemas e Industrial
Bogotá, Colombia
2018

Un hombre libre en nada piensa menos que en la muerte, y su sabiduría no es una meditación de la muerte, sino de la vida

B. Spinoza

Agradecimientos

El estudio de sistemas de cómputo ha tenido un desarrollo si se puede decir factorial en las últimas décadas, el uso de dispositivos con capacidades de cálculo , almacenamiento y red es tan común como hace unos años tomar notas, hacer cálculos con papel y lápiz, incluso el acceso a este tipo de dispositivos es sencillo , es por ello que el presente trabajo nace de la iniciativa de tener un acceso justo a recursos y aplicaciones de cómputo , esta visión esta centrada en el trabajo desarrollado por varios años de mi director el profesor Jorge Eduardo Ortiz Triviño, a quien agradezco su guía, paciencia y consejo,logrando plasmar sus ideas y las mías en el proyecto del Grupo TLÖN.

No son suficientes estas páginas para nombrar a todos los que me han apoyado,mi familia con su paciencia, apoyo y esfuerzo, mis padres quienes me han dado lo mejor de sí, mis hermanos quienes confían en mi y mi novia siempre presente en los momentos alegres , duros y complejos de este trabajo, como no nombrar a mis compañeros del grupo de investigación TLÖN quienes con sus ideas e interpretación de este sistema han logrado crear de forma novedosa resolver los problemas e integrar los componentes de cómputo y modelos de este sistema, del mismo modo mis compañeros del grupo de Servidores y Servicios (SYS) de la Oficina de Tecnologías de la Universidad Nacional quienes con su apoyo y confianza han aportado de manera singular a este trabajo, del mismo modo a la profesora Cristiana Bolchini mi tutora en el Politécnico de Milán, un paso importante para cerrar el proceso de desarrollo e implementación de este proyecto y a mi gran amigo Antonio Miele, investigador y profesor del Politécnico de Milano, del mismo modo el apoyo técnico y formal del profesor Oscar Agudelo y el ingeniero Juan Carlos Rivera por su experiencia y enseñanzas a lo largo de este trabajo, y por supuesto a Dios.

Resumen

Las redes de comunicaciones dinámicas estocásticas como las redes ad hoc, están inmersas en ecosistemas altamente distribuidos como lo es Internet, incluso es un medio para implementar tecnologías como Ciudades inteligentes, Internet de las Cosas (IoT), entre otros. Estos ambientes distribuidos donde la cantidad de recursos de cómputo disponibles, la calidad de servicio (QoS), y la naturaleza de servicios solicitados por los usuarios son factores determinantes en las interacciones hombre-máquina/ máquina-máquina, requieren de una abstracción que permita identificar y definir las interacciones entre los miembros de estos sistemas para ejecutar las tareas distribuidas con el fin de obtener el servicio sin importar las limitaciones de los dispositivos, una forma de resolver este problema es la construcción de un sistema operativo virtualizado orientado a redes ad hoc, bajo premisas sociales como la justicia o la equidad, necesaria en estos ambientes computacionales cambiantes.

Palabras clave: Red Ad hoc, Virtualización Inalámbrica, Sistema Distribuido, Computación Paralela, Algoritmos Distribuidos, Orquestación, Estado, Bigrafos.

Abstract

The dynamic stochastic communication networks such as ad hoc networks are immersed in highly distributed ecosystems such as the Internet, it is even a means to implement technologies such as Smart Cities, Internet of Things (IoT), among others. These distributed environments where the amount of computing resources available, the quality of service (QoS), and the nature of services requested by users are determining factors in human-machine / machine-machine interactions, require an abstraction to identify and define the interactions between the members of these systems to execute the distributed tasks in order to obtain the service regardless of the limitations of the devices, one way to solve this problem is the construction of a virtualized operating system oriented to ad hoc networks, under social premises such as justice or equity, necessary in these changing computational environments

Keywords: Ad hoc Networks, Wireless Virtualization, Distributed System, Parallel Computation , Distributed Algorithms, State, Bigraphs.

Contenido

Agradecimientos	VII
Resumen	IX
Lista de símbolos	xx
1. Introducción	1
1.1. Objetivos	3
1.1.1. Objetivo General	3
1.1.2. Objetivos Específicos	3
1.2. Justificación	4
1.3. Aportes al conocimiento	5
1.4. Producción Académica	7
2. Sistemas Distribuidos	10
2.1. Teorema CAP	10
2.1.1. Consistencia	11
2.1.2. Disponibilidad	12
2.1.3. Tolerancia a Particiones	12
2.1.4. Evolución del Teorema CAP	12
2.2. Teorema del Consenso	13
2.3. Programación Distribuida	14
2.3.1. Modelo de Programación Distribuida	16
2.4. Sistemas Operativos Distribuidos	17
3. Gestión de recursos virtualizados en redes Ad hoc	19
3.1. Generalidades	19
3.2. Marco Conceptual	22
3.2.1. Virtualización Ligera	25
3.3. Orquestadores	26

4. Modelamiento Formal del Sistema	30
4.1. Modelado con Bigrafos	30
4.2. Reglas de Reacción	36
5. Computación Social	38
5.1. Estado del Arte	38
5.1.1. Modelo Estado -PseudoEstado	41
5.1.2. Inmanencia	42
5.1.3. Justicia	43
5.1.4. Estado	43
5.1.5. Modelos de Pseudo Estado Propuestos	44
5.1.6. Leyes Naturales y Leyes Positivas	46
6. Proyecto Sistema TLÖN	47
6.1. Formalización de los Principios Sociales	50
6.2. Modelo Social Inspirado Propuesto - Capa Virtualización inalámbrica	53
7. S.O.V.O.R.A.	54
7.1. Primeras interpretaciones	54
7.2. Estructura del Sistema y Microservicios	55
7.3. Modelo de la solución	57
7.4. Esquema de la Virtualización	58
7.5. Modelo del Sistema Operativo	59
7.5.1. Red Ad hoc -Pseudo Territorio	60
7.5.2. Diagrama Capa de Virtualización	61
7.5.3. Diagrama Orquestador	62
7.5.4. Diagrama Agente Local	62
7.5.5. Diagrama SOVORA	64
7.5.6. Primitivas del Sistema Virtualizado	65
7.5.7. Instituciones del Sistema	68
7.5.7.1. Generador de Instituciones	68
7.5.7.2. Dispersión y Recopilación	71
7.5.7.3. Agente Móvil	72
7.5.7.4. Balanceador de Carga	73
7.5.7.5. Seguridad Virtualización	74
7.5.7.6. Sistemas de Archivos	75
7.5.7.7. Detector de Fallas	77
7.5.7.8. Socket para Sistemas Distribuidos	79
7.5.8. Dispositivos de Entrada Salida	79

7.5.9. Sistema Multi Agente	81
7.6. Descripción de las Pruebas	83
7.7. Componentes Conceptuales del modelo propuesto	83
7.8. Modelo de la solución	84
7.9. Modelo de Comunicación	85
7.10. Modelo de Aplicaciones	89
7.11. Política Desarrollada	91
8. Pruebas y Resultados	96
8.1. Caracterización de los Dispositivos	96
8.1.1. Comunicación modo Ad hoc	100
8.1.2. Caracterización Modelo de Energía	100
8.2. Comparación Cualitativa de la Solución Propuesta	105
8.3. Resultados de Operación de la Solución	108
8.3.1. Escenario 1 - Monarquía	115
8.3.2. Escenario 2 - República Presidencialista	118
8.3.3. Escenario 3 - República Parlamentaria	121
9. Conclusiones y Recomendaciones	130
9.1. Conclusiones	130
9.2. Recomendaciones y Trabajo futuro	133
A. Redes Ad hoc	134
A.1. Clústers en Redes Ad hoc	136
A.2. Nubes Móviles	137
A.2.1. Nubes Sociales	138
A.3. Internet de las Cosas (IoT)	139
A.4. Protocolos de Enrutamiento en MANETS	144
A.4.1. Protocolo B.A.T.M.A.N	146
A.4.1.1. Tipos de Páquetes B.A.T.M.A.N	146
B. Anexo: Modelamiento Algebraico de Sistemas Distribuidos	148
B.1. Cálculo II	149
B.1.1. Definiciones	149
B.1.2. Semántica operacional	151
B.2. Bigrafos	153
B.2.1. Definiciones	154
B.2.2. Operaciones Básicas	156

C. Anexo:Técnicas de Diseño y Análisis Estructurado- SADT	159
D. Anexo: Instalación de la Solución	162
E. Anexo: Caracterización de Energía Odroid XU4	171

Lista de Figuras

2-1. Teorema CAP. Adaptado de Lynch (1996)	11
3-1. Virtualización en Sistemas de Cómputo	21
3-2. Virtualización de Redes Liang y Yu (2015)	21
3-3. Virtualización de Redes Inalámbricas Wen y cols. (2013)	22
3-4. Virtualización de LTE. Liang y Yu (2015)	23
3-5. Proyecto GENI.Liang y Yu (2015)	24
3-6. Contenedor de Docker	26
3-7. Vista de Alto Nivel de un Orquestador de Sousa y cols. (2018)	27
3-8. Orquestador TLÖN	28
3-9. Clasificación de NSO de Sousa y cols. (2018)	29
4-1. Bigrafo del sistema Estado 0	31
4-2. Grafo de Lugar e Hipergrafo	32
4-3. Bigrafo Estado 1	34
4-4. Bigrafo Estado 2	35
4-5. Bigrafo Estado 3	35
4-6. Bigrafo Estado 4	36
5-1. Algoritmos Socio Culturales Kumar y cols. (2018)	40
5-2. Socio Inspiración en ICT Ferscha y cols. (2012)	41
5-3. Taxonomía Social Inspiración Bio inspiración	42
5-4. Modelo de Estado. Adaptado de Hobbes (2001)	44
6-1. Principios de la Social inspiración.	48
6-2. Sistema TLÖN	48
6-3. Modelo Social Inspirado.	51
6-4. Modelo Sartreano.Sartre (1946)	52
7-1. Microservicio Newman (2015)	56
7-2. Modelo Propuesto para SOVORA	58

7-3. Modelo Virtualización	59
7-4. Niveles de S.O.V.O.R.A.	60
7-5. Red Ad hoc Pseudo Territorio	61
7-6. Modelo Virtualización Inalámbrica	62
7-7. Diagrama Orquestador	63
7-8. Políticas Orquestador	63
7-9. Agente Local	64
7-10. Diagrama S.O.V.O.R.A.	65
7-11. Modelo Instituciones	69
7-12. Modelo Agente Móvil	72
7-13. Modelo Balanceador	73
7-14. Modelo Seguridad Propuesto (CIA)	74
7-15. Modelo de Disco Distribuido	76
7-16. File System espacio de Usuario	78
7-17. Detector de Fallas propuesto	79
7-18. Modelo de Funcionalidad de dispositivos I/O Silberschatz y cols. (2014)	80
7-19. Modelo de Funcionalidad de dispositivos I/O Silberschatz y cols. (2014)	81
7-20. Modelo Sistema Multi Agente Flasiński (2016)	82
7-21. Orquestador y Agente Local	85
7-22. Modelo de comunicación	86
7-23. Modelo de comunicación Agente Local Aplicaciones	87
7-24. Modelos de Aplicaciones en Contenedores	91
7-26. Tablas política Orquestador	92
7-25. Política Observar - Decidir - Actuar	92
8-1. Modelo de Energía RPI 3	103
8-2. Modelo de Energía RPI zero	103
8-3. Modelo de Energía Odroid XU 4	105
8-4. Escenario de Pruebas	108
8-5. Modelos de Pseudo Estado implementados	109
8-6. Consola Web SOVORA	112
8-7. Consola de Comandos SOVORA	112
8-8. Agente Local SOVORA	113
8-9. Orquestador SOVORA	113
8-10. Log Agente Local SOVORA	114
8-11. Log Orquestador SOVORA	114
8-12. Modelo del escenario	115
8-13. Control de Ancho de Banda	116
8-14. Energía Consumo de energía en un nodo	116

8-15. Throughput en de las aplicaciones en cada nodo	117
8-16. Throughput de las aplicaciones en un nodo	117
8-17. Modelo del escenario	118
8-18. Control de los nodos	120
8-19. Modelo del escenario	121
8-20. Resultados de operación del escenario	121
8-21. Resultados de operación del escenario nodo 1	122
8-22. Resultados de operación del escenario nodo 2	123
8-23. Resultados de operación del escenario nodo 3	124
8-24. Resultados de operación del escenario nodo 4	125
8-25. Control de los nodos	126
8-26. Control de los nodos	127
8-27. Media de recurso CPU vista desde el orquestador	128
A-1. Red Ad hoc	135
A-2. Clúster Red Ad hoc	136
A-3. Arquitectura IoT	140
A-4. Protocolos de Enrutamiento en Manets	145
A-5. Protocolo Batman	147
B-1. Servidor de impresión (Izq. antes de la interacción, Der. Después de la interacción) $(\bar{b}a.S b(c).\bar{c}d.P) \xrightarrow{\tau} (S \bar{a}d.P)$ Milner (1999)	151
B-2. Modelo cliente servidor Cálculo II Varela y Agha (2013)	151
B-3. Bigrafo adaptado de Milner (2008)	153
B-4. Bigrafo Abstracto Milner (2008)	154
B-5. Bigrafo con controles Milner (2008)	155
B-6. Bigrafos Elementales adaptado de Milner (2009)	156
B-7. Ion Discreto Milner (2009)	157
B-8. Anatomía de un Bigrafo Milner (2009)	158
C-1. Diagramas SADT.	159
C-2. Formas SADT Tlön.	161

Lista de Tablas

4-1. Reglas de Reacción	36
4-2. Forma Algebraica de las Reglas de Reacción	37
5-1. Tipos de Estado	45
5-2. Formas de Gobierno	45
8-1. Arquitectura de los dispositivos usados	97
8-2. Aplicaciones seleccionadas para la realización de las pruebas	97
8-3. Resultados algoritmo CRC32 - MiBench RPi 3	98
8-4. Resultados algoritmo CRC32 MiBench Rpi zero	98
8-5. Resultados algoritmo CRC32 MiBench Odroid	98
8-6. Resultados algoritmo SHA MiBench RPi 3	98
8-7. Resultados algoritmo SHA MiBench RPi zero	99
8-8. Resultados algoritmo SHA MiBench Odroid	99
8-9. Resultados algoritmo Jpeg MiBench RPi 3	99
8-10. Resultados algoritmo Jpeg MiBench RPi zero	99
8-11. Resultados algoritmo Jpeg MiBench odroid	99
8-12. Resultados prueba Ad hoc batctl/ batman-adv RPi 3	100
8-13. Resultados prueba Ad hoc batctl/ batman-adv RPi zero	100
8-14. Resultados prueba Ad hoc batctl/ batman-adv Odroid	101
8-15. Throughput ODROID Frecuencia Little 600 MHz XU-4	101
8-16. Consumo de Energía RPi 3 con Wifi	101
8-17. Consumo de Energía RPi 3 sin Wifi	102
8-18. Consumo de Energía RPi zero con Wifi	102
8-19. Consumo de Energía RPi zero sin wifi	102
8-20. Comparación Cualitativa con otras soluciones similares en el mercado	107
8-21. Escenarios Validados	110
8-22. Análisis Estadístico Figura 8-16	118
8-23. Análisis Resultados nodo 1	122
8-24. Análisis Resultados nodo 2	123

8-25. Análisis Resultados nodo 3	124
8-26. Análisis Resultados nodo 4	125
8-27. Análisis Estadístico Orquestador	128
A-1. Modelo de Tabla en Protocolos Reactivos Loo y cols. (2016)	145
E-1. Odroid Little sin Wifi Frecuencia 400MHz.	171
E-2. Odroid Little sin Wifi Frecuencia 600MHz.	171
E-3. Odroid Little sin Wifi Frecuencia 1000MHz.	172
E-4. Odroid Little sin Wifi Frecuencia 1600MHz.	172
E-5. Odroid Little sin Wifi Frecuencia 1800MHz.	172
E-6. Odroid Little sin Wifi Frecuencia 2000MHz..	173
E-7. Odroid Little con Wifi Frecuencia 400MHz..	173
E-8. Odroid Little con Wifi Frecuencia 600MHz..	173
E-9. Odroid Little con Wifi Frecuencia 1000MHz..	174
E-10. Odroid Little con Wifi Frecuencia 1600MHz..	174
E-11. Odroid Little con Wifi Frecuencia 1800MHz..	174
E-12. Odroid Little con Wifi Frecuencia 2000MHz.	175

Lista de símbolos

Abreviaturas

Abreviatura	Término
3G	Tercera generación de tecnología celular
4G	Cuarta generación de tecnología celular
5G	Quinta generación de tecnología celular
ABM	Agen Base Model
ACL	Agent Communication Language Specifications
ACP	Algebra of Communicating Systems
AL	Agente Local
ALFRED	Almighty Lightweight Fact Remote Exchange Daemon
AMS	Agent Management System
ANA	Autonomic Network Architecture
ANM	Autonomic Network Management
AP	Access Point Wireless
ARP	Address Resolution Protocol
ASCII	American Standard Code for Information Interchange
ATM	Asincronous Transfer Mode
AWS	Amazon Web Services
B.A.T.M.A.N.	Better Approach to Mobile Ad hoc networking
BDI	Belief, Desire and Intention.
BGP	Border Gatwway Protocol
BRS	Biographical Reactive System
CAP	Consistency Availavility Partition tolerance
CC	Cloudlet Computing
CCS	Calculus of Concurrent Systems
CIA	Consistency Integrity Availability
CSP	Communicating Sequencial Processes
DAO	Data Access Object

Abreviatura	Término
DDOS	Distributed Denial Of Service
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Service
DSCP	Differentiated Service Code Point
FC	Fog Computing
FIPA	Foundation for Intelligent Physical Agents
FSF	Free Software Foundation
GNS 3	Real time network simulator
GNU GPGNU	General Public License
GoS	Grade of Service
GSM	Global System for mobile communications
IDS	Intrusion Detection System
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IPS	Intrusion Prevention System
IPv4	Internet Protocol Versión 4
IPv6	Internet Protocol Versión 6
ISO	International Organization for Standardization
ISP	Internet Service Provider
ITIL	Information Technology Infrastructure Library
ITU	Telecommunication Standardization Sector
JADE	Java Agent Development Framework
JRE	Java Runtime Environment
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LGPL	Lesser General Public License Versión 2
LLC	Logical Link Control
LLDP	Link Layer Discovery Protocol
LTE	Long Term Evolution
MA	Mobile Agent
MAC	Media Access Control
MANET	Mobile Ad hoc network
MAS	Multi Agent System
MEC	Mobile Edge Computing
MPLS	Multi Protocol Label Switching
MRTG	Multi Router Traffic Grapher

Abreviatura	Término
MSS	Maximum Segment Size
MST	Multiple Spanning Tree
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NFV	Network Function Virtualization
NFS	Network File System
NMS	Network Management System
NS 3	Network simulator 3
OGM	Originator Message B.A.T.M.A.N.
ONF	Open Network Foundation
OR	Orchestrator
OSI	Open System Interconnection
OSPF	Open short path first
PC	Personal computer
PEPA	Performance Evaluation Process Algebra
PRS	Procedure Reasoning System
QoS	Quality Of Service
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request For Comments
SDN	Software Define Network
SFTP	Secure File Transfer Protocol
SNMP	Simple Network management protocol
SOA	Service Oriented Architecture
S.O.V.O.R.A.	Sistema Operativo Virtualizado Orientado a Redes Ad hoc
SON	Self organizing network
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
UAM	Ubiquitous Abstract Machine
UDP	User Datagram Protocol
UUID	Universal Unique Identifier
VLAN	Virtual LAN
VM	Virtual Machine
VPN	Virtual Private Network
WAF	Web Application Firewall
WCCP	Web Cache Control Protocol
WNV	Wireless Network Virtualization

Capítulo 1

Introducción

Las dinámicas humanas son objeto de estudio en sociología, psicología, filosofía, al igual que en otras ramas de las ciencias sociales , consisten en contar el interés suscitado en el arte, los deportes y en general en todas las disciplinas y contextos humanos. La literatura es un ejemplo de ellos, cientos historias provienen de la mente de los hombres, mundos posibles y eternos son creados como es el caso de TLÖN Borges y Clemente (1956), un mundo dominado por las leyes de la psicología y no de la física , una invención de un grupo de sabios liderados por un genio oculto; este modelo de mundo abstracto, ideal y regido por leyes no convencionales lo hace parte de una ficción, pero ¿acaso el hombre imaginó la conquista del espacio, la corriente eléctrica, los sistemas ubicuos o eran parte de una ficción?, es la ciencia y la investigación quienes posibilitan el desarrollo e interpretar como nos relacionamos en medio de un ambiente cambiante, como interpretamos la metáfora de estar vivos y aprender a escribir sobre el lienzo del mundo, sobre las páginas de la historia, como releer a grandes hombres invisibles en su tiempo pero gigantes en los nuestros.

La capacidad de ver el mundo y sus interacciones de forma diversa permiten formular soluciones a problemas inconclusos o mejorar la forma de resolver los actuales, recordemos unas palabras de Thomas Kuhn donde la búsqueda en últimas es encontrar el cambio de paradigma a través de la innovación, “E incluso hoy en día, para cambiar una vez más de terreno, parte de la dificultad con que nos encontramos para ver las profundas diferencias que median entre la ciencia y la tecnología se relaciona con el hecho de que ambos campos poseen como atributo obvio el progreso” Kuhn (2012). En este sentido el progreso que a simple vista es bienestar debe estar ligado al hecho de nuestra humanidad, la capacidad de reconocer al otro y entender la dimensión de ser seres sociales.

En este aspecto el presente trabajo toma como cimientos la social inspiración, basado en conceptos comunes en ciencias sociales como lo son la justicia Rawls (2012), que también ha sido tomada como modelo base en sistemas de cómputo para distribución de recursos, la inmanencia concepto clave en el mundo de Spinoza Spinoza (s.f.), y el modelo de Estado de Thomas Hobbes Hobbes (2001) en su

magnánima obra Leviatan, para construir un sistema de cómputo socialmente ideal gobernado por agentes artificiales sobre un medio dinámico, estocástico, descentralizado, en esencia una red Ad hoc Sarkar y cols. (2007); Zhao y Jain (s.f.); Ramanathan y cols. (2010); Correa y cols. (s.f.); Feeney y Nilsson (2001), este es el medio perfecto para probar un sistema social artificial, o mejor un pseudo estado, dadas las condiciones del nuevo ecosistema de redes, con la aparición de conceptos como IoT computación de borde, micro datacenter, fog computing entre otras ,Negash y cols. (2018); Bonomi y cols. (2014); Raychaudhuri y Mandayam (2012); Skarlat y cols. (2017). Es necesario entonces encontrar una forma diferente de interpretar estas interacciones entre diversos equipos, redes y usuarios inter o hyper conectados Xiao y Pan (2009), puede considerarse una ficción, pero a si mismo como lo fue la idea de inteligencia artificial o los modelos bio inspirados, corresponde a una línea que siempre ha estado abierta, desde el deseo de hacer máquinas que imiten a los hombres. Hoy día podemos afirmar que las máquinas son una extensión de cada ser humano, ya sea para hacer un cálculo, desarrollar una teoría, simular un modelo o entender como se formó el universo, pero ¿ lograran estas estar a la par del ser humano?.

El sistema propuesto es una construcción por capas: la primera la infraestructura que es la red ad hoc, la segunda la virtualización inalámbrica Liang y Yu (2015) la cual es propuesta en este trabajo, compuesta por dos elementos fundamentales un Agente Local (AL) en cada nodo miembro y un orquestador o gestor de las interacciones entre los miembros de la red, esto con el fin de desplegar un sistema multi agente Wooldridge (2009) sobre esta abstracción donde comunidades de agentes ideales despliegan servicios distribuidos, de manera transversal existe un lenguaje de programación de propósito específico Van-Roy y Haridi (2004) el cual permite manipular todo este conjunto de abstracciones, desarrollado bajo las interacciones de los agentes, las capas y los usuarios.

Este trabajo esta desarrollado de la siguiente manera, en el capítulo A se hace un recorrido de los conceptos de la primer capa de nuestro modelo las redes ad hoc , el capítulo 2 introduce sobre la naturaleza de los sistemas distribuidos, el capítulo 3 aborda el desarrollo de virtualización de redes, virtualización inalámbrica y como es su inclusión en los ambientes móviles y ubicuos, el capítulo B toma como base los aportes del profesor Robert Milner en el construcción de la teoría de Bigrafos para dar el sustento formal a esta investigación, este modelo permite modelar sistemas ubicuos reactivos, teniendo en cuenta todo este marco conceptual y formal se introduce en el capítulo 5 la evolución de la social inspiración en computación y el modelo propuesto, en el capítulo 6 se da un recorrido por el proyecto denominado Sitema TLÖN base de este trabajo, el capítulo 7 se presentan los diseño y elementos que constituyen el sistema y los resultados obtenidos con la implementación de este modelo en sistemas embebidos, ejecutando los modelos de pseudo estado propuestos; Los resultados de esta implementación se presentan en el capítulo 8 finalmente el capítulo 9 contiene las conclusiones del trabajo realizado y el trabajo futuro propuesto.

Es claro que en ciencia y en investigación no existe una respuesta a todos los problemas, pero en cambio es válido proponer una, aunque esta no sea la más elocuente, sin embargo el objeto de este trabajo es ser el abono de nuevas investigaciones en modelos social inspirados y redes de telecomunicaciones dinámicas y estocásticas.

1.1. Objetivos

El presente trabajo está enmarcado dentro del proyecto principal del grupo de investigación llamado TLÖN (Grupo de Investigación en Redes de Telecomunicaciones Dinámicas y Lenguajes de Programación Distribuidos), que pretende construir un modelo de cómputo distribuido con énfasis en el modelo social inspirado.

1.1.1. Objetivo General

Objetivo General

Construir un subsistema del Sistema TLÖN que implemente virtualización inalámbrica para gestionar recursos de procesamiento, memoria (RAM), almacenamiento y dispositivos de E/S distribuidos en una red Ad Hoc, mediante el modelo de pseudo estado.

1.1.2. Objetivos Específicos

Objetivos Específicos

1. Construir el subsistema de virtualización inalámbrica para gestionar recursos de cómputo.
2. Construir el subsistema de virtualización inalámbrica para gestionar recurso de red y almacenamiento.
3. Construir el subsistema de operación base de las pseudo sociedades de agentes mediante la simulación de instituciones de un estado (Sistema Operativo), para un sistema de cómputo distribuido.
4. Validar los subsistemas en una red Ad Hoc, mediante la definición de tres escenarios de prueba.

1.2. Justificación

Las características de autoconfiguración y auto-organización propias de las redes Ad Hoc Basagni y cols. (2004); Loo y cols. (2016) hacen de ellas una de las opciones para implementar redes sin infraestructura, además de poseer cualidades de autonomía, la opción de ser escalables, al igual que la inclusión de agentes para gestionar los posibles comportamientos pseudosociales, comportamientos emergentes y las variaciones dinámicas en su estructura, permiten construir un modelo de cómputo con la capacidad de realizar transacciones y tareas distribuidas sobre una ambiente poco amigable como lo es el inalámbrico.

Hoy en día es común tener necesidades de servicios ubicuos y junto con la proliferación de redes y servicios en Internet, se hace necesario plantear un escenario híbrido y escalable que permita la interacción de agentes intra e inter redes, el concepto de nube móvil Fitzek y Katz (2013) toma un significado más inmediato al momento de diseñar e implementar servicios y aplicaciones en ecosistemas complejos como IoT o Fog Computing. Una de las respuestas posibles es generar abstracciones lógicas más complejas Zhou y cols. (2015); Wen y cols. (2013); Liang y Yu (2015); Portnoy (2012) dentro del ecosistema con las limitaciones de recursos y canales de comunicación, de forma sencilla pero con la capacidad de prestar disponibilidad, integridad y tolerancia a particiones.

El modelo propuesto puede evaluarse fuera de los modelos tradicionales y tratarse desde el punto de vista de dinámicas sociales que permitan abordar los problemas de asignación de recursos , distribución de tareas de forma consecuente con la cantidad de recursos de cada dispositivo.

1.3. Aportes al conocimiento

Aportes de este Trabajo

- a) El primer aporte es el esquema de mapeo propuesto a través del modelo Agente Local y Orquestador, sobre una red Ad hoc con protocolo de enrutamiento proactivo.
- b) El segundo aporte es la realización de un modelo formal bajo el modelo de Bigrafos propuesto por el profesor Robin Milner , el cual describe las interacciones entre las redes Ad hoc y el modelo virtualizado propuesto.
- c) El tercer aporte es el desarrollo del modelo social inspirado TLÖN en sistemas distribuidos inalámbricos, mediante la generación de políticas locales y globales que regulan las interacciones de los miembros del sistema.
- d) El cuarto aporte es el desarrollo de un artefacto de software que permite la virtualización inalámbrica, en un sistema de cómputo distribuido como lo es una red Ad hoc.
- e) El quinto aporte es la implementación del sistema virtualizado, bajo la arquitectura de microservicios en dispositivos embebidos heterogéneos.
- f) El sexto aporte es el desarrollo de un esquema social inspirado modular para el despliegue de un sistema operativo distribuido denominado S.O.V.O.R.A.
- g) El séptimo aporte es la integración de diferentes módulos del Sistema TLÖN, desarrollados en otras tesis del grupo de investigación, manteniendo la visión de la arquitectura para la realización del prototipo base del Sistema de Cómputo Distribuido TLÖN.

1.4. Producción Académica

Contribución Académica

a) Cuatro Artículos de investigación.

- Towards the design of a social-inspired module for the decision making of nodes in an ad hoc network Sistemas & Telemática González, J. E., Ortiz, J. E., & Ceballos, H. Z. (2016). Towards the design of a social-inspired module for the decision making of nodes in an ad hoc network. *Sistemas & Telemática*, 14(36), 9-25.
- Protocol Conversion Approach to Include Devices with Administration Restriction on a Framework of Reference of Management Network WEA 2017: Applied Computer Sciences in Engineering García, M. T., Zarate, H., & Triviño, J. O. (2017, September). Protocol Conversion Approach to Include Devices with Administration Restriction on a Framework of Reference of Management Network. In *Workshop on Engineering Applications* (pp. 72-83). Springer, Cham.
- Toma de Decisiones en Nodos del Sistema Distribuido TLÖN Ingeniería y Región González Ortiz, J. (2017). Toma de Decisiones en Nodos del Sistema Distribuido TLÖN. *Ingeniería Y Región*, 17, 61-71. <https://doi.org/10.25054/22161325.1525>
- SNMP converter and the forward data collection as management method in dynamic distributed networks"WEA 2018: Applied Computer Sciences in Engineering "García, M. T., Zarate, H., & Triviño, J. O. (2018, October). SNMP converter and the forward data collection as management method in dynamic distributed networks . Springer, Cham.

b) Cuatro Participaciones en eventos académicos internacionales.

- Sistema de Telecomunicaciones Social-Inspirados Mediante Comunidades de Agentes. CICOM- 5^o CONGRESO INTERNACIONAL DE COMPUTACIÓN MÉXICO COLOMBIA CICOM 2015
- Model in an ad-hoc network for data acquisition in sensor networks Colcom 2015 Colombian Conference on Communications and Computing
- Internet de las Cosas y los sistemas de producción agropecuarios: oportunidades para la academia y la industria SIMPOSIO DE INGENIERÍAS Y BIOTECNOLOGÍA: TECNOLOGÍAS APLICADAS AL SECTOR AGROPECUARIO
- Modelo para la adquisición de datos de soporte a la toma de decisiones en cadenas logísticas mediante redes tipo Ad-Hoc II Congreso Internacional de Industria y Organizaciones

c) Participación en el Docker-Con Europa Copenhagen 2017

Contribución Académica

- d) Dos Cursos de divulgación en Sistemas Embebidos y Redes de Sensores, avalados por Colciencias.
 - Curso en redes de Sensores sobre sistemas embebidos, orientado a agrónoma
 - Curso en redes de Sensores
- e) Una pasantía Nacional en la Universidad Nacional de Colombia- Oficina de Tecnologías de la información y las comunicaciones Sede Bogotá.
- f) Una pasantía internacional en el Politécnico de Milán - Grupo De Arquitectura de Sistemas de Cómputo.
- g) Apoyo al proceso de Categorización del Grupo de investigación TLÖN- Grupo de Investigación en Redes de Telecomunicaciones Dinámicas & Lenguajes de Programación Distribuidos, finalizado en reconocimiento y categoría C de Colciencias.
- h) Desarrollo de un libro Simulación de Redes Ad hoc - Wireless Ad hoc Networks Simulation using ns-3. (En revisión).
- i) Profesor asistente de Métodos numéricos y sistemas Operativos (3 semestres).
- j) Creación del Semillero de Investigación en Seguridad de la Información - UQBAR,avalado como Grupo de Estudio GET por Bienestar de Ingeniería.
- k) Desarrollo de ocho propuestas de tesis de Maestría como apoyo a este trabajo.
 - Contents management algorithm for Ad Hoc networks bio- inspired in the quorum sensing utilized by gram negative bacteria. Jorge Ernesto Parra Amaris. (Finalizada)
 - Construcción un sub-sistema de software que permita generar movilidad de agentes artificiales de acuerdo con el rol que ellos desempeñen sobre el Sistema TLÖN cuyos recursos se comparten a través de una red Ad hoc. German Dario Alvarez Rodriguez. (Finalizada)
 - Implementación de un módulo de software que dote a nodos potenciales de una red ad hoc con las capacidades para ingresar, permanecer, gestionar recursos y salir de un sistema distribuido que opera sobre esa red. John Edwar Gonzalez. (En espera de Jurados)
 - Construcción de un modulo de seguridad Informática en el sistema TLÖN que permita tener características de disponibilidad. Santiago jose Molina. (En Desarrollo)
 - Construcción de un protocolo de comunicaciones para el sistema TLÖN que permita la inclusión y monitoreo de dispositivos con limitaciones de gestión en un marco de referencia de administración de red. Edgar Mauricio Tamayo. (Espera de Jurados)

Contribución Académica

- j) Desarrollo de ocho propuestas de tesis de Maestría como apoyo a este trabajo.
- Construcción de un módulo de control virtualizado para realizar balanceo de carga en MANETs. Oscar David Rueda Dimate . (En Desarrollo)
 - Módulo de seguridad de la información, social-inspirado y orientado hacia el objetivo de integridad en MANETs dentro del contexto del sistema de cómputo distribuido TLÖN. Manuel Leoned Molano Saavedra (En Desarrollo)
 - Construcción de un módulo para el Kernel de Linux que permita la gestión de nodos dentro de sistemas distribuidos o IoT. Edward Camilo Carrillo Estupiñan. (Proyecto aceptado para maestría.)
- k) Desarrollo de tres trabajos de grado (Pregrado) como parte de este trabajo.
- Implementación de un clúster de red Ad-Hoc para la adquisición de datos en la coordinación de redes logísticas. Julian Felipe Latorre Ochoa. (Finalizada)
 - Diseñar, desarrollar y puesta en producción de un sistema distribuido que permita la recolección, análisis y presentación de la información de variables de la red física de la Universidad Nacional- Sede Bogotá, permitiendo generar una función de densidad representada en forma de mapa de calor. Rodrigo Ivan Espinel Villalobos.(Finalizada)
 - Implementación de una tarea colaborativa entre dos robots móviles usando una arquitectura distribuida sobre una red ad hoc. Yosef Esteban Ramirez Rosero. (En Desarrollo)
- l) El desarrollo de Software S.O.V.O.R.A. que permite realizar virtualización inalámbrica sobre redes Ad hoc.

Capítulo 2

Sistemas Distribuidos

Uno de los grandes problemas dentro de las arquitecturas y estructuras presentadas hasta ahora, es el manejo de los datos, como controlar y mejorar la operación de un sistema, el cual posee un comportamiento distribuido. Es importante indicar que un sistema distribuido es aquel que opera en diferentes estaciones, es decir los recursos se encuentran en diferentes máquinas, donde el usuario como tal es inconsciente de estas interacciones Haddad y cols. (2013). Estas características distribuidas traen consigo una variedad de retos, para el manejo de información y conocer el estado completo del sistema, uno de los elementos claves es identificar con claridad que servicios requieren Consistencia en la información, Disponibilidad y Tolerancia a particiones Lynch (1996).

Bajo esta característica se presentará el Teorema CAP (Consistency, Availability, Partition Tolerance), esto con el fin de diseñar un sistema que mitigue en cierto grado los inconvenientes comunes en este tipo de sistemas en las transacciones realizadas, la solución propuesta no está exenta de estos problemas pero busca tener un grado de confiabilidad al momento de realizar las operaciones básicas para su despliegue.

2.1. Teorema CAP

A partir de la conjetura de Eric Brewer en el año 2000, donde afirmó “Es imposible para un servicio web proporcionar las siguientes tres garantías: **Consistencia, Disponibilidad y Tolerancia a particiones**”, esta conjetura más tarde fue abordada por los profesores Seth Gilbert y Nancy Lynch Gilbert y Lynch (2002), quienes analizaron y formalizaron la conjetura de Brewer, para postular el Teorema CAP y como consecuencia el Teorema del Consenso.

Aunque esta conjetura está basada en servicios web, es necesario aclarar, que estos servicios están en red, las transacciones realizadas sobre las aplicaciones y bases de datos asociadas son eventos comu-

nes en el despliegue de redes ad hoc y de sistemas distribuidos. Un ejemplo claro es la tolerancia a particiones, ¿cómo se comporta un sistema cuando pierde conectividad?, estas y más preguntas son respondidas en el teorema CAP y donde claramente se muestra la incapacidad de entregar estos tres elementos en un sistema distribuido y se debe elegir dos de las tres características para ser cumplidas y garantizadas por un sistema distribuido.



Figura 2-1: Teorema CAP. Adaptado de Lynch (1996)

2.1.1. Consistencia

El modelo formal aborda inicialmente los objetos de datos atómicos, este modelo garantiza la transacción correcta dentro de un sistema distribuido. Un ejemplo es la solicitud de memoria compartida, si existe la característica de consistencia es posible ver las transacciones de un sistema distribuido como si fueran un único nodo Gilbert y Lynch (2012).

La Consistencia es la respuesta correcta a la consulta realizada, dependiendo del tipo de servicio se hace necesario dar unas restricciones o dar prioridad a esta característica, una clasificación inicial de servicios es la siguiente:

1. **Triviales:** Servicios donde no se requiere coordinación entre servidores
2. **Consistencia Débil:** Mantiene los servicios funcionales pero no es un requerimiento obligatorio para el despliegue de servicios.

3. **Simples:** Al existir la necesidad de tener el estado de cada una de las transacciones, se convierte en un factor determinante una corrección directa de los servicios, esto es en últimas la semántica del servicio, es decir, cuáles son las especificaciones secuenciales y las operaciones de carácter atómico.
4. **Complicados:** Esta clase de servicios no puede ser especificadas por acciones secuenciales, en las tareas de lectura /escritura.

2.1.2. Disponibilidad

La disponibilidad es la capacidad de responder cada una de las peticiones de un nodo sin falla dentro de un sistema distribuido, esto es en esencia, la terminación de cualquier operación o algoritmo usado por el sistema de forma inesperada dentro del tiempo de operación. Existen límites dentro de la definición de disponibilidad, por ejemplo el tiempo de duración de la tarea o la llegada de la respuesta. Podemos decir entonces, que la disponibilidad es la capacidad de entregar una respuesta a cada petición realizada.

2.1.3. Tolerancia a Particiones

Finalmente la tolerancia a particiones, contempla la posibilidad de que la red podría permitir en algunos escenarios la pérdida de mensajes enviados entre los nodos. Generando grupos de servidores que se comunican en diferentes partes del sistema, cuando este evento ocurre existen varios segmentos del sistema que se comunican y continúan realizando transacciones entre los segmentos o grupos formados, al perderse el mensaje puede verse como la separación de los nodos al momento de enviar un mensaje.

2.1.4. Evolución del Teorema CAP

Existen dos escenarios claves para ser analizados, las redes síncronas y las asíncronas, en este trabajo, tenemos redes dinámicas y estocásticas, las redes ad hoc, ¿Cómo se abordaría este fenómeno en este tipo de redes?.

En redes móviles inalámbricas Gilbert y Lynch (2012), al ser poco confiables, entran en la órbita del teorema CAP, en el factor de tolerancia a particiones, el tiempo en que duran estos cambios de grupos semi estables en el sistema, se tienen fallas de operación, latencia y perdida de paquetes, sin contar

con los diferentes procesos ejecutados en sistemas que requieren operar con bases de datos distribuidas, es por ello que en la siguiente sección se abordara una forma de manejar la inestabilidad de los procesos en sistema distribuidos.

2.2. Teorema del Consenso

Teniendo en cuenta lo mencionado en la sección anterior el teorema CAP presenta retos en el diseño de sistemas distribuidos, y más aún en sistemas estocásticos y dinámicos como las redes ad hoc, al agregar este factor a un sistema distribuido, o mejor de base de datos distribuida la latencia se incluye como un elemento adicional en la conformación de un sistema virtualizado como el propuesto en este trabajo, en respuesta al manejo de sistemas asíncronos y la imposibilidad de resolver escenarios donde se efectúan transacciones que involucran la estabilidad del sistema y dependen puramente de la actualización de estos datos, para mitigar estos escenarios es propuesto el teorema del consenso Fischer y cols. (1985), teniendo en cuenta a su vez el contexto expuesto en los problemas bizantinos Lamport y cols. (1982), el objetivo de formular protocolos de consenso es determinar el dato correcto entre un grupo de nodos, mediante el envío de mensajes e intercambio de información entre los miembros con el fin de reducir las fallas en el sistema y mantener la consistencia de la información enviada, esto con el objetivo de conocer el estado local y global del sistema. Este protocolo formulado para sistemas asíncronos, tiene en cuenta el estado del proceso si esta terminado o en operación, el conjunto de mensaje enviados entre nodos y la toma de decisión de los mismos para determinar la acción subsecuente o la definición de los estados locales y globales del sistema.

Es importante definir que cada sistema es representado por un grafo construido a partir de los estados definidos en el sistema y por un conjunto de n procesos que toman decisiones a partir de los estados reportados, se tiene en cuenta los ciclos de decisión infinita al igual que las reglas de ambivalencia de los sistemas, para la demostración formal de este protocolo.

En sistemas distribuidos uno de los inconvenientes cuando hay grandes flujos de información de control o de datos, es el manejo de registros o mensajes duplicados en el sistema, el modelo propuesto de múltiple consenso Ongaro y Ousterhout (2014), permite distribuir en mayor medida los elementos clave que componen las reglas de consenso para garantizar la consistencia de la información.

Protocolo de Múltiple Consenso

El protocolo cuenta con tres roles para su operación

- **Líder Fuerte:** El protocolo propone una forma de liderazgo fuerte para simplificar la gestión de registros duplicados. Un ejemplo es un flujo único de registro del líder a los demás servidores.
- **Elección de Líder:** Este protocolo usa tiempos aleatorios para la elección de líderes, con el fin de resolver la elección con tiempos de reloj más cortos y de forma simple.
- **Cambios de pertenencia:** Los mecanismos definidos para el cambios en el conjunto de servidores en un clúster usa un consenso conjunto donde las mayorías de dos configuraciones diferentes se superponen durante transiciones dentro del sistema.

Bajo este modelo el algoritmo inicia con la elección del líder, quien replica los estados a través de todo el sistema, finalmente gestiona los registros de forma segura administrándolos como máquinas de estado finitas bajo cinco acciones posibles: primero la elección segura del líder, segundo sólo el líder puede agregar datos, tercero el líder valida si hay registros repetidos, cuarto la integridad es garantizada cuando el líder hace la actualización del registro en el sistema y por último la seguridad de la máquina de estados. Es decir, sólo se puede aplicar una entrada de registro en un servidor dado un índice en la máquina de estados, dado este registro ningún otro servidor puede modificarlo. De esta forma el sistema es controlado y se mantiene la integridad y consistencia de la información, este protocolo es visto como uno de los más sobresalientes protocolos de consenso en sistemas distribuidos.

2.3. Programación Distribuida

Uno de los modelos más cercanos a la implementación de un lenguaje de propósito general para sistemas distribuidos con gestión de recursos de cómputo es Petabricks Ansel y cols. (2009). Este lenguaje y compilador permite validar el rendimiento de algoritmos a pesar de que maneja los mismos problemas de ejecución como datos distribuidos, paralelismo, transformaciones y bloqueos. Este es un lenguaje orientado a programación paralela, es decir se obtiene una resolución paralela del problema buscando el mejor algoritmo para el problema seleccionado y alcanzando la granularidad deseada por el usuario. El compilador de este lenguaje permite autoconfigurar los algoritmos para lograr alcanzar la mejor solución al problema en cuestión.

Este lenguaje tiene como constructores principales dos elementos las *transformaciones* y las *reglas* Van-Roy y Haridi (2004), La transformación es análoga a una función, este elemento permite hacer la elección del algoritmo que puede ser llamado desde otra transformación, código escrito en otros lenguajes

o desde la linea de comandos, el header contiene los argumentos, desde donde se invoca y mediante el cual se ejecuta el algoritmo.

Las reglas por su parte definen como se computa una sección del código, con el fin de hacer un avance hacia la solución del problema, estas reglas a su vez tienen dependencias directas y parametrizadas por el compilador, dadas por variables libres. Del mismo modo hacen referencia a cuales son los elementos de entrada y cuales sus salidas, junto con el compilador delimitan la aplicación de reglas en ciertas porciones del código a ejecutar, así como a capacidad de variar al granularidad y conocer los estados intermedios del lenguaje. Dadas estas características de paralelismo y auto configuración, este lenguaje permitiría la administración de múltiples procesadores, sin embargo no contiene aún el concepto de Sistema Distribuido, ni las necesidades de virtualización, sin embargo es un referente en cuanto a la integración procesador, compilador, algoritmo, vital en este tipo de investigaciones Henz y cols. (1993).

Un primer acercamiento a estos sistemas puede derivar en una Taxonomía de Sistemas Distribuidos; en esta primera aproximación el sistema esta compuesto por tres tipos de recursos procesador, memoria y red, los dos primeros como recursos compartidos y el tercero como el enlace de estos recursos. Un elemento clave en estos sistemas y derivados del acceso a Internet es la conectividad, el cual modifica el modelo de computación de alto rendimiento, pasando al de computación colaborativa, el segundo abre las puertas a modelos sociales donde la colaboración es base de las relaciones humanas, al existir esta apertura dos problemas son adicionados a los mencionados en la sección anterior (Teorema CAP), el problema de Seguridad y Nombres dentro de los servidores y miembros de la red.

A pesar de que la propiedad de Alta Disponibilidad es deseada en un sistema de cómputo, la gestión de recursos dados por el Clúster definidos por la probabilidad $A(t)$ de que el clúster entregue el servicio solicitado en un instante t de tiempo dado, es deseable tener un probabilidad uniforme con el fin de mantener una calidad de servicio equitativa entre los miembros del clúster.

Finalmente los modelos de distribución los podemos ver de la siguiente forma:

- a Multiprocesadores y Memoria Compartida. (Cluster Computing).
- b Memoria y Procesador Distribuido - Adicionan distribución al sistema.(Cluster Computing).
- c Memoria y procesador Distribuido con Fallas Parciales- Adicionan Fallas de conectividad. (Cluster Computing).
- d Sistemas Distribuidos Abiertos- Agregan Colaboración (Internet Computing).

2.3.1. Modelo de Programación Distribuida

Para manejar los problemas descritos en la sección anterior es importante recordar dos conceptos uno los puertos, donde inician las interacciones en la Red y el otro las celdas , las cuales contienen estados parciales o globales de las entidades del sistema. Un Lenguaje que maneja esta tareas de distribución es Mozart Van Roy (2005).

Por un lado los estados pueden ser manejados como hilos, de otro lado el almacenamiento de flujos de datos, puertos, celdas son localizados como procesos dentro de las entidades del lenguaje. Para realizar interacciones entre los miembros del sistema distribuido y de lo procesos involucrados aparece una primera aproximación de un modelo orientado a procesos, las cuales son entidades clasificadas de la siguiente forma:

- a **Estado Completo:** (Hilos, celdas, puertos, objetos) tienen un estado interno. El manejo de estos estados internos es una parte sensible para mantener un estado global coherente con el sistema, estos valores introducen restricciones a los comportamientos posibles del sistema.
- b **Asignación simple:** (Variable de flujos de datos y streams): tiene una operación fundamental la Unión. Esta operación permite unir todas las referencias posibles de una variable de flujo de datos en el sistema, convirtiéndose en la primera etapa de coordinación del mismo.
- c **Sin Estado:**(procedimientos, funciones, registros, clases) son valores que no necesitan tener referencias a procesos porque son constantes.

Una forma de declarar los estados y comportamientos globales del sistema distribuido es referenciando las entidades a procesos, sin embargo en el caso de los puertos no se hace referencia a procesos porque al igual que una celda que almacena valores muy puntuales del sistema un puerto es una unidad básica del modelo distribuido. Teniendo en cuenta la operación asíncrona y las interacciones de la semántica de un lenguaje distribuido, es posible ver los comportamientos de las entidades como consecuencia de los comportamientos de las partes que lo componen.

La declaración de un subconjunto de estados del modelo concurrente o el estado precedente, es el soporte de funcionamiento del modelo distribuido en un tiempo futuro, este sistema se puede denominar abierto si un proceso se puede conectar independientemente con otros procesos corriendo sobre una computación distribuida en tiempo de ejecución, sin conocer de primera mano cuales son los procesos con los que puede interactuar o intercambiar información. Este modelo agrega un nuevo interrogante ¿Cómo evitar las confusiones cuando una máquina o computación independiente se comunica con otras?, existen varios modelos para hacerlo en este trabajo nombraremos cuatro.

- a **Referencias:** Es un medio de acceso ineludible para acceder a cualquier entidad del sistema.
Ejemplo: un hilo puede ser referenciado a una entidad dentro de otro proceso.

- b **Nombres:** un nombre es una constante incluyente que es usada para implementar abstracciones de datos seguras. **Ejemplo** autenticación en sistemas, identidad inicial y tokens.
- c **Tickets:** Es un medio global para acceder a cualquier entidad del sistema. Es una cadena que puede ser transmitida bajo cualquier protocolo para dar información del estado global del sistema.
- d **URL(Uniform Results locator):** Es una referencia global a un archivo, este codifica el hostname del web server y la ubicación del archivo en esa máquina, las URLs son usadas para intercambiar información persistente entre procesos.

2.4. Sistemas Operativos Distribuidos

Para poder indicar que es un sistema operativo distribuido ó, multi-procesador es necesario indicar que es un sistema distribuido. Se puede decir que un sistema distribuido es *una colección de computadores independientes que para el usuario son mostrados como un simple computador*, dos elementos son contrastados en esta definición por una parte el hardware que es independiente y autónomo, y por otra parte la abstracción de software que es visto como un solo componente por el usuario, otro componente fundamental es *la comunicación*, esta capacidad permite que las máquinas conozcan los estados globales, coordinadores y los nodos o equipos con los que puede interactuar.

En este sentido un sistema operativo distribuido debe ofrecer transparencia en la ubicación, la migración de recursos y aplicaciones, la replicación de los servicios existentes, la propiedad de concurrencia para poder compartir los recursos y no menos importante el paralelismo de las actividades fundamentales del sistema. Una de las abstracciones más usadas en sistemas distribuidos es el hilo (*thread*), este artefacto permite dividir las tareas y la carga en diferentes equipos, como procesos aislados identificados o pueden ser usados como un recurso particular denominado *pool thread* (Conjunto de hilos), estas rutinas permiten la paralelización y la capacidad de generar comportamientos reactivos como enviar un mensaje de respuesta.

Asociado a este esquema se encuentra el problema de ubicación de recursos, de no ser controlado se generan cuellos de botella o sobre cargas de tareas en un nodo o equipo en particular, de esta manera es necesario definir políticas tanto de ubicación como de transferencia dentro del sistema , con el fin de balancear la carga y el consumo de recursos de nodos específicos. Este modelo permite generar estrategias para la gestión de los recursos y tener un esquema de programación de tareas (*scheduler*), para optimizar la respuesta de los dispositivos hacia las solicitudes de los usuarios.

Uno de los desafíos de los sistemas operativos es la operación en tiempo real, el manejo de este tipo de operación diverge desde el código, por ejemplo el código de tiempo real interactúa con sensores u

otros elementos para su operación, el software de tiempo real permite una perdida de conectividad para su operación con una probabilidad asociada, en cambio en el hardware de tiempo real esto es inaceptable porque podría generar consecuencias como catástrofes o la perdida de vidas, en consecuencia existen sistemas intermedios en los cuales se pueden detener procesos o mantener fuera de línea rutinas sin consecuencias fatales. En este sentido los protocolos de comunicación, las estrategias de programación de tareas y las políticas deben responder de forma dinámica a cambios en el ambiente, a la cantidad de recursos disponibles y modelos de detección de fallas.

Estos elementos se condensan en cierto modo en los esquemas de virtualización (Pesada y Ligera) y la operación de sistemas en la nube (Cloud Computing), los cuales son tratados de manera más detallada en el capítulo 3.

Capítulo 3

Gestión de recursos virtualizados en redes Ad hoc

3.1. Generalidades

Dos de los principales elementos de la virtualización son la *abstracción* y la *representación* van de Belt y cols. (2017), para describirlos podemos dar la siguiente explicación, dentro del concepto de abstracción esta incluida la forma en la cual los sistemas interactúan y la complejidad de cada una de las interacciones, del mismo modo en la descripción de la abstracción se debe dejar de lado detalles que no son relevantes al momento de definirla. En cambio la representación , contiene la instanciación, concepto clave para describir las relaciones entre las entidades abstractas y el mundo físico, existe toda una Teoría de la Representación útil para definir estas relaciones entre el dominio físico P y el dominio abstracto M , esta relación se hace mediante un mapeo de la forma $R : p \rightarrow m_p$, donde el objeto abstracto m_p se dice ser una representación abstracta del objeto físico p .

En cuanto al dominio abstracto podemos dar la siguiente relación como evolución o proceso $C : m_p \rightarrow m'_p$ donde cambia un objeto abstracto m_p en otro objeto abstracto m'_p , análogamente el cambio se realiza sobre el objeto físico mediante un cambio de estado.

En este sentido la virtualización reside en el mundo abstracto, ya que en el mundo físico no se puede variar los dispositivos de cualquier forma, el sentido de la virtualización reside en los niveles de representación y la jerarquía de los mismos, aunque básicamente ocurre en este dominio, es importante diferenciar en que nivel ocurre y que tratamiento debe darse a los recursos abstractos de cada nivel de representación, ya que no es solo allí donde se puede crear la abstracción virtualizada, a pesar de la representación de recursos reales del mundo físico, la virtualización es el mapeo de recursos los cuales pueden ser alterados en alguna dimensión, los recursos virtuales generados son en esencia los mismos que los reales pero reservados para ese propósito, por ello los recursos mapeados pueden ser explotados,

combinados y distribuidos de diversas maneras.

El mapeo es el mecanismo de virtualización Goldberg (1973) que esta definido por una función f que mapea el conjunto de recursos virtuales \mathcal{V} , al conjunto de recursos reales \mathcal{R} , los recurso virtuales pueden ser el dominio de f y los reales el codominio, la ecuación 3-1 representa la función de mapeo.

$$f : \mathcal{V} \rightarrow \mathcal{R} \cup \{t\} \quad (3-1)$$

El mapeo puede representarse de diversa formas, para el objeto de este trabajo el mapeo es realizado *muchos a muchos*, donde se mapea múltiples recursos virtuales a múltiples recursos reales, la combinación de estos recursos mapeados, crean recursos adaptados a las necesidades del usuario o aplicación desplegada. Efectos deseados en el mapeo son la recursión y el aislamiento para controlar la competencia de recursos y evitar interferencias entre los procesos desplegados en los recursos reales distribuidos.

Definiciones

Abstracción: Es el acto de ignorar detalles al considerar características generales. Sin perder información importante del fenómeno.

Abstracto: Algo que existe como pensamiento, idea, concepto pero sin existencia física.

Representación: el acto de simbolizar o describir algo de una forma particular.

Instanciación: Crear una representación completa de un concepto o idea.

Podemos definir entonces la virtualización como la creación de un conjunto de arquitecturas lógicas usando un conjunto dado de entidades físicas, pero de una manera transparente para el usuario. Desde hace mucho tiempo la virtualización ha sido usada no solo en computación, también en redes como se ve en la figura 3-2, por ejemplo, con el uso de redes privadas virtuales (VPN) o redes de área local virtuales(VLAN) por nombrar unos casos. Con la aparición de la computación en la nube y la expansión de Internet se requieren de nuevas arquitecturas para su despliegue, incluso la heterogeneidad de dispositivos, es allí donde la virtualización inalámbrica se convierte en una de las tecnologías más promisorias para el Internet del Futuro, en contraste un recuento de estas tecnologías habilitadoras en sistemas de cómputo se puede observar en la figura 3-1Liang y Yu (2015); Mouradian y cols. (2016); Xiao y Pan (2009).

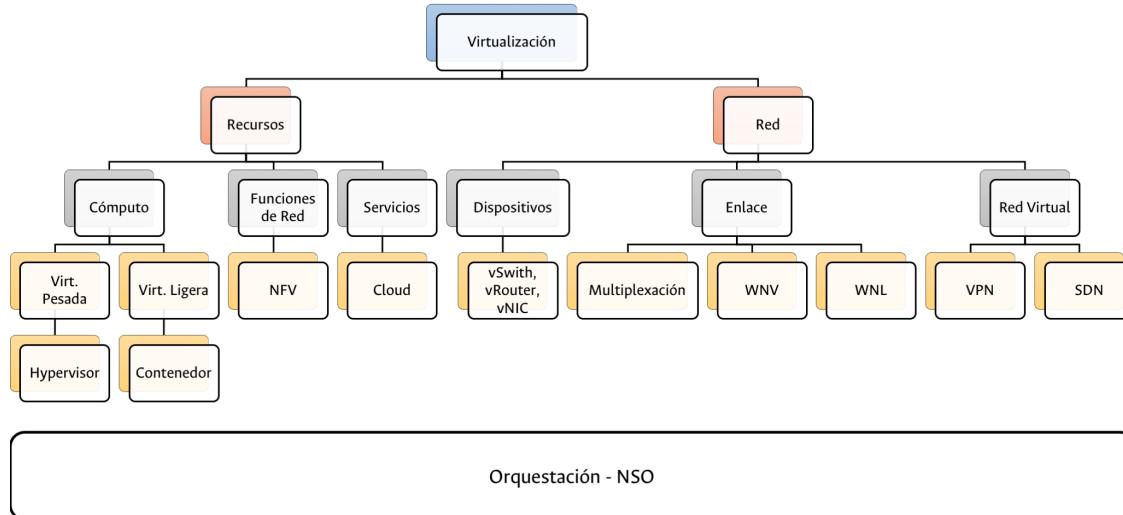


Figura 3-1: Virtualización en Sistemas de Cómputo

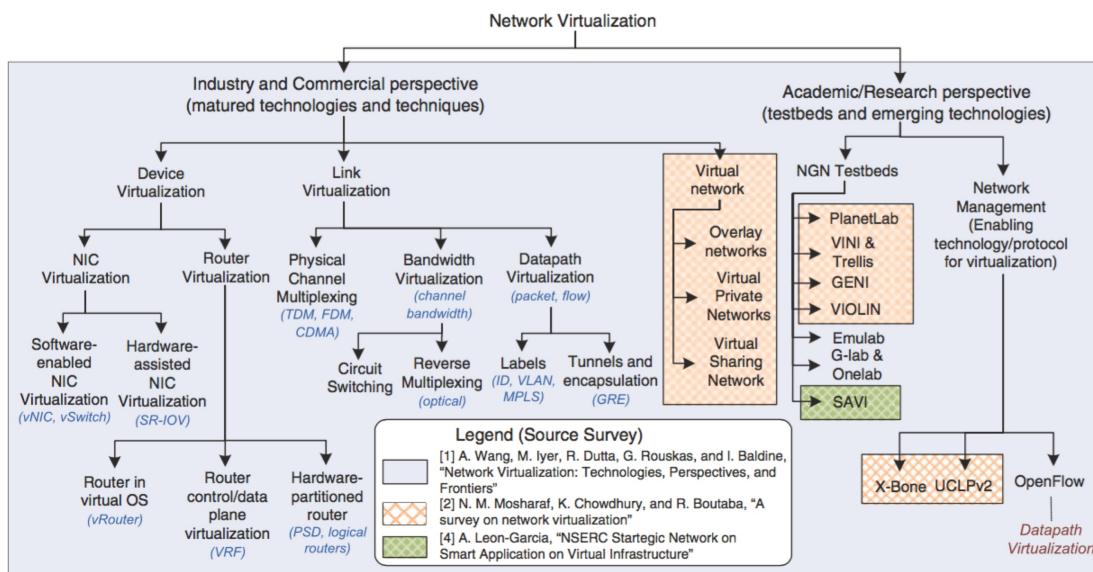


Figura 3-2: Virtualización de Redes Liang y Yu (2015)

La característica de movilidad y las necesidades de convergencia han llevado consigo el desarrollo en paralelo de diversas tecnologías orientadas a infraestructura como servicio (SaaS en inglés) o incluso

redes como servicio (NaaS en inglés), todo esto enmarcado en sistemas de virtualización, o mejor en esquemas donde el software o una capa lógica superior cobran fuerza para su despliegue como lo son las redes definidas por Software (SDN en inglés).

Las redes definidas por Software (SDN), son una arquitectura emergente de red donde el control es separado del plano de datos y es directamente programable, es considerada la tecnología más promisoria para implementar redes virtuales, especialmente en redes de control Zhou y cols. (2015), posee elementos claves en la virtualización inalámbrica como: la separación del plano de control del de datos, un controlador centralizado y vista general de la red, interfaces abiertas entre los dispositivos en el plano de control y también en el plano de datos y programación de la red para aplicaciones externas, es importante indicar que Openflow no es SDN, es un estándar administrado por la Open Networking Foundation (ONF), es un protocolo que permite la interacción entre las capas de control y datos de la arquitectura SDN Wen y cols. (2013), esta separación no implica virtualización. Un esquema de virtualización de SDN puede observarse en la figura 3-3.

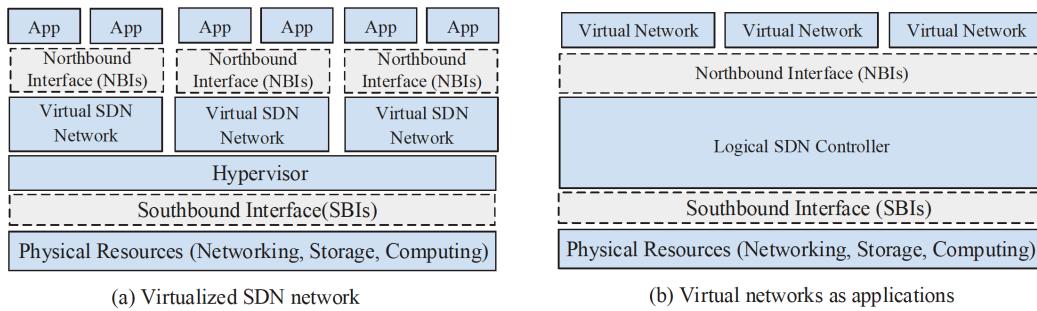


Figura 3-3: Virtualización de Redes Inalámbricas Wen y cols. (2013)

Por su parte la Virtualización de Funciones de red (NFV), descompone un conjunto dado de servicios de red en un conjunto de funciones de red que pueden ser implementadas sobre un hardware adecuado mediante técnicas de virtualización, se puede decir que NFV es un forma de virtualización de nodos, por ejemplo el controlador de movilidad de red, el servidor subscriptor local son elementos que pueden ser virtualizados sobre NFV.

3.2. Marco Conceptual

Existen diferentes perspectivas de virtualización de redes, sin embargo, tanto en la académica como en la industrial como se evidencia en la figura 3-2, relaciones en los temas clave y en los productos obtenidos, la generalización consiste en obtener virtualización de recursos y la administración de los

dispositivos involucrados en la red, pero a su vez se generan desarrollos que benefician a ambas partes en la implementación de prototipos y creación de nuevas tecnologías: unas emergentes, obtenidas en la academia y otras maduras obtenidas en la industria, a pesar de esto la virtualización inalámbrica es un punto en común donde ambos están aportando para generar un marco conceptual completo y una aplicación robusta.

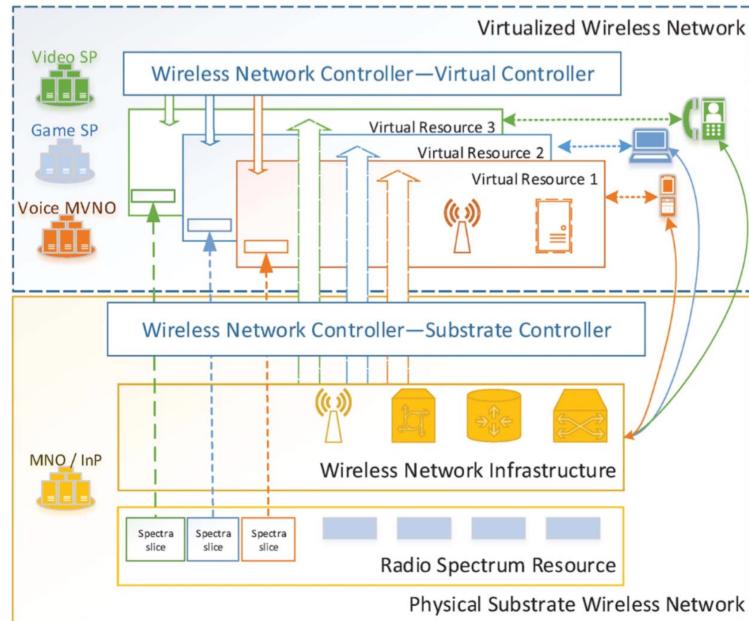


Figura 3-4: Virtualización de LTE. Liang y Yu (2015)

Teniendo en cuenta lo anterior se han centrado esfuerzos de virtualización en redes celulares, específicamente en las tecnologías 5G, donde se usa el esquema de hypervisor, el cual también es manejado por la máquinas virtuales, este capa ubicada entre el hardware y el sistema operativo permite la abstracción de recursos y generar la capa base del sistema distribuido, en la figura 3-4 se observa el esquema de virtualización de un eNode de LTE (Long Term Evolution), donde se crean nodos virtuales dentro de la infraestructura para administrar el espectro que en este caso, se convierte en un recurso limitado y escaso, adicionalmente se lleva a cabo la virtualización de los recursos físicos.

Es importante resaltar que a pesar de que existen proyectos especializados en virtualización inalámbrica Liang y Yu (2015) como GENI visto en la figura 3-5, el cual utiliza la abstracción de contenedor para administrar los recursos distribuidos, contiene un framework de virtualización donde se incluye la autenticación, las librerías, y una infraestructura para la realización de pruebas o experimentos específicos para el desarrollo de prototipos y arquitecturas de red.

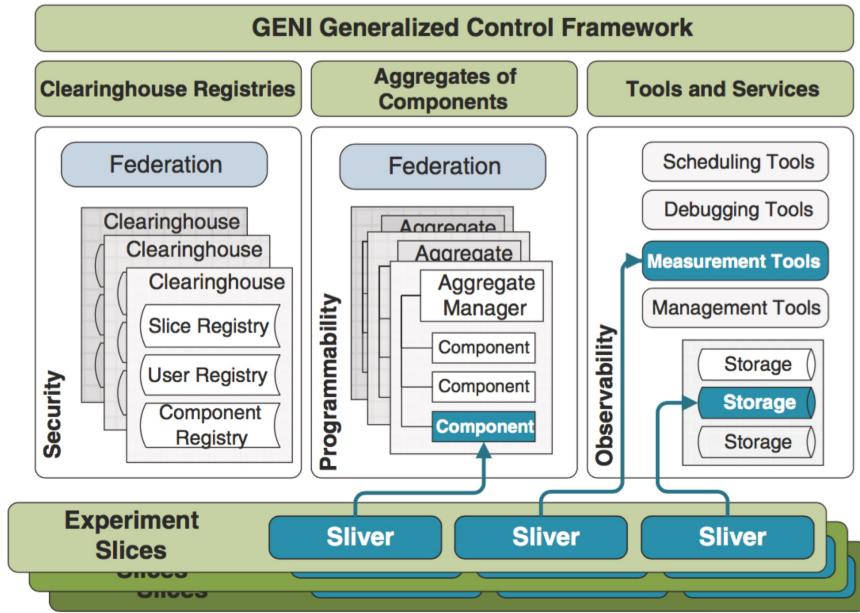


Figura 3-5: Proyecto GENI.Liang y Yu (2015)

Otro framework de virtualización es Planet Lab Peterson y Roscoe (2006), un proyecto nacido en el 2002 donde se incorpora el concepto de virtualización distribuida como la extensión geográfica de una red, por medio del cual el usuario crea un escenario de prueba en el cual la infraestructura es proporcionada por los nodos miembros de la red de Planet Lab. Al no existir un backbone dedicado las conexiones son realizadas sobre internet, este sistema crea máquinas virtuales locales en los dispositivos miembros y eleva la abstracción de recursos a un nivel superior. A diferencia de GENI este usa la virtualización sobre Linux Vservers, cada uno con un hypervisor o Virtual Machine Monitor (VMM), realizando la administración sobre las particiones y los privilegios entregados al agente que administra el VMM; la comunicación al sistema principal PlanetLabCentral (PLC) garantiza el acceso a los recursos para las pruebas diseñadas por el usuario.

Estos proyectos de virtualización inalámbrica no están bajo las condiciones de las redes Ad hoc, las cuales en cierto modo son opuestas a las características mínimas requeridas por la virtualización inalámbrica: coexistencia, flexibilidad, aislamiento, escalabilidad, convergencia, algunas son similares como movilidad y el problema de recursos limitados como lo es el espectro. ¿Es posible realizar Virtualización inalámbrica sobre Redes Ad hoc ? Uno de los aspectos importantes es definir que recurso se va a virtualizar y con que técnica, inicialmente veremos como es posible virtualizar el enlace de Red inalámbrico (WLV en inglés), elemento de entrada para la WNV.

WLN - Virtualización de Enlace Inalámbrico

Es el proceso de virtualizar enlaces inalámbricos creando recursos virtuales que son aislados y los cuales pueden ser usados por diferentes tecnologías y /o configuraciones independientes. Es diferente del término recurso compartido ya que la virtualización hace posible la combinación de recursos. Dos de los desafíos inmersos en este concepto son la movilidad y las condiciones del medio, debido a interferencias, potencia y el continuo broadcast sobre la red. Como objetivos dentro de la WNV se tienen: Zhang y cols. (2017) Combinar / Compartir Recursos, provisionar Calidad de Servicio QoS, Realizar una gestión flexible de los recursos proporcionados y tener servicios novedosos sobre la red. En contra parte como desafíos tenemos el control de interferencia de canal adyacente y co-canal, adaptación a los cambios de la red tanto físicas como lógicas(tráfico, recursos) y no menos importante la heterogeneidad multi dimensional de dispositivos desde el núcleo de la red, las tecnologías de radio acceso y e general los recursos de red.

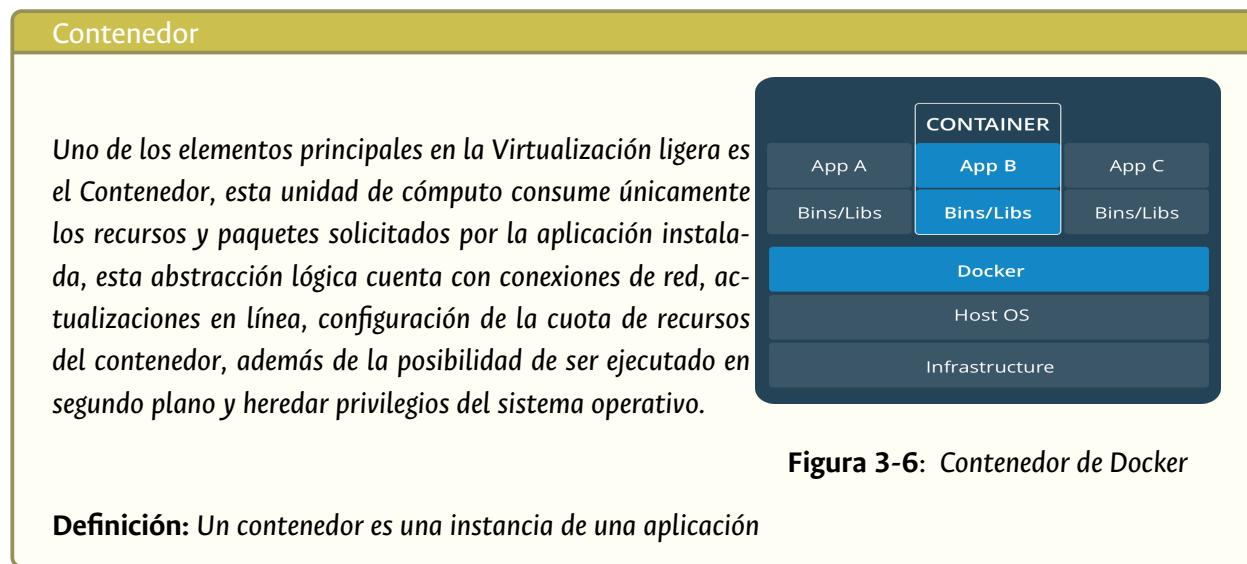
3.2.1. Virtualización Ligera

La necesidad de gestionar mejor los recursos de cómputo en ambientes dinámicos, elásticos e hyper conectados requieren un manejo diverso de los recursos y obviar en algunos casos capas adicionales como el hecho de tener un sistema operativo nativo y uno huésped para poder desplegar servicios. Uno de los modelos de virtualización ligera, la cual busca mitigar la dependencia entre el software y hardware en la virtualización tradicional con la inclusión de la abstracción de hypervisor o también conocido como Monitor de Máquinas Virtuales (VMM en inglés), este modelo esta orientado a la creación de Máquinas Virtuales (VM) dónde cada máquina virtual posee su propio sistema operativo Chelladhurai y cols. (2017).

La evolución de los ecosistemas Matthias y Kane (2015) de cómputo, la inclusión de tecnologías ubicuas, sistemas con recursos limitados y aplicaciones multi usuario hace necesario introducir una herramienta que permita encapsular los procesos creados en artefactos distribuidos para cualquier tipo de aplicación, permitiendo diferenciar las etapas de diseño, pruebas y producción, agilizando las actualizaciones y gestionando los flujos de trabajo de los grupos de desarrollo.

Bajo estas premisas es desarrollado Docker como un modelo de computación basado en el concepto de Contenedor, con similitudes a una máquina virtual y aislamiento del OS, pero con la particularidad de usar únicamente los recursos, librerías y piezas de software necesarias para el despliegue de la aplicación, Miell y Sayers (2016); Mouat (2015), además de eso permite el fácil despliegue, control y monitoreo de las aplicaciones desarrolladas y aisladas en un contenedor, capacidades multiusuario, migración y replicación.

Adicionalmente esta herramienta puede ofrecer dentro de su operación el manejo de los paquetes de software dependiendo de las habilidades de los desarrolladores, unificar en una imagen los filesystems y paquetes de aplicaciones necesarios para desplegar un servicio, usar paquetes para probar y entregar el mismo artefacto computacional a todos los sistemas operativos oficiales en todos los ambientes, abstraer aplicaciones de software desde el hardware sin sacrificar recursos. Bajo estas premisas se pueden desplegar imágenes en diferentes ambientes de operación y gestionar aplicaciones con recursos limitados, suministrando un ambiente propicio para las redes ad hoc y el entorno de este trabajo.



3.3. Orquestadores

Para hablar de orquestadores en primera instancia se debe hablar de redes, telecomunicaciones y de servicios en la nube, tres tecnologías convergen para el desarrollo de Orquestadores de Sousa y cols. (2018), estos sistemas son : i)las Redes Definidas por Software (SDN en inglés), ii)las Funciones Virtuales de Red (NFV en inglés) y iii)la Virtualización de Recursos en la nube. Estas tecnologías habilitadoras de servicios distribuidos y aplicaciones ubicuas, requieren de un mecanismo de organización que permita optimizar el uso de los recursos y prestar servicios en las condiciones deseadas, el cruce de estas tecnologías se puede observar en la figura 3-7

Por su parte SDN propone desacoplar el plano de control del plano de datos, con esta arquitectura propuesta los enruteadores y switches hace envíos a elementos lógicos de la red proporcionados por una entidad externa, denominada Sistema de operación de Red (NOS en inglés), este controlador per-

mite generar abstracciones que desplieguen servicios orquestrados en los niveles lógicos y virtuales, permitiendo de esta forma la automatización.

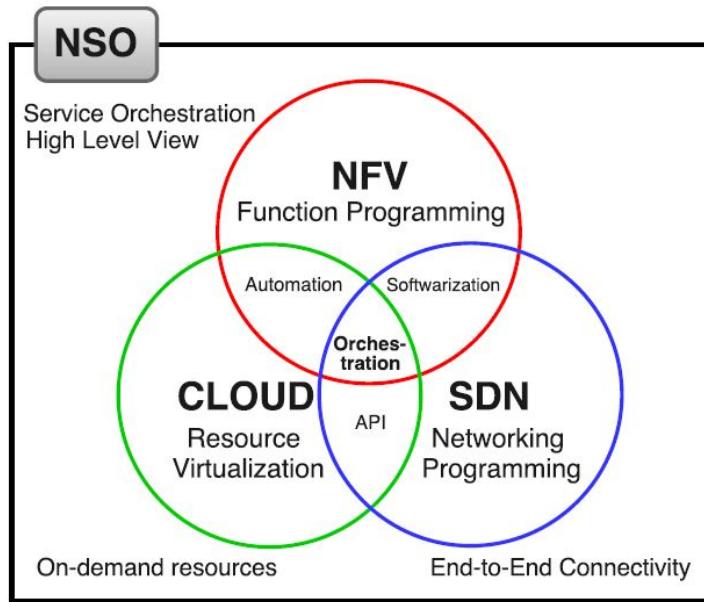


Figura 3-7: Vista de Alto Nivel de un Orquestador de Sousa y cols. (2018)

En este mismo sentido las NFV son responsables de separar las funciones de red desde el hardware y ofrecerlo mediante servicios virtualizados o en servidores de propósito general Gao y cols. (2017); Giotis y cols. (2015); Sun y cols. (2017), generando Funciones de Red Virtualizadas (VNF en inglés), este despliegue de funciones requiere menos hardware, incluyendo más abstracciones de software con el fin de gestionar mejor el tráfico.

Finalmente la computación en la nube permite modelar accesos ubicuos o por demanda a redes y la capacidad de compartir recursos, ser provisionados de forma rápida y automática, en este sentido el cliente solo paga por lo que usa. Los modelos de servicios en la computación en la nube se dividen en tres categorías Software como Servicio (SaaS), Plataforma como Servicio (SaaS), e infraestructura como Servicio (IaaS), los servicios de orquestación involucran el despliegue dinámico, la administración y mantenimiento de servicios en las plataformas mencionadas según las necesidades de los clientes Santoro y cols. (2017); Asnaghi y cols. (2016).

Ahora bien, el término orquestación está presente en varias disciplinas como la música, multimedia, computación en la nube, SDN entre otras, pero la analogía es el conjunto de instrumentos que tocan juntos una sinfonía, el trabajo de cada uno de los instrumentos toma sentido cuando los otros se unen

en completa sincronía para obtener la melodía deseada, se puede indicar que un orquestador puede coordinar tareas dentro de un sistema en diferentes capas, manejar flujos de tráfico y definir las acciones dependiendo de las interacciones entre las aplicaciones y el usuario, es decir, hay una diferencia entre automatizar y orquestar. En el campo de aplicaciones web por ejemplo, el proceso de orquestar puede ser definido como la ejecución de un proceso que puede interactuar con los procesos y/o servicios internos y externos, de forma dinámica, flexible y adaptable a los cambios para mantener la continuidad de negocio, la lógica de ejecución de actividades y la automatización.

Orquestador

La orquestación también está presente en sistemas biológicos entre los límites de la auto-organización y de la aparición de comportamientos emergentes, algunos orquestadores comerciales en virtualización y gestión de redes son Kubernetes, Docker Swarm, AWS Auto Scaling Groups, Mesos Marathon y Vrealize Operation Manager, en este sentido hay cuatro ejes de control o automatización de procesos, los pre diseñados, los orquestados, los auto organizantes y los emergentes , para efectos de este trabajo el enfoque es en el eje de orquestación.

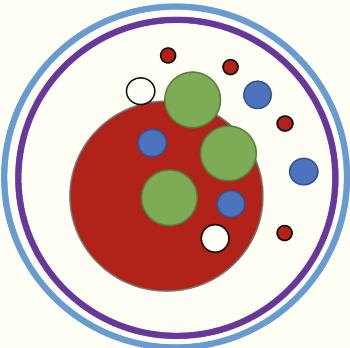


Figura 3-8: Orquestador TLÖN

Definición: Un orquestador es una instancia o agente que gestiona todas las interacciones de un clúster

En general se considera la orquestación Kang y cols. (2010); Zaalouk y cols. (2014) como un mecanismo de automatización de tareas y mecanismo de control de los recursos y elementos que soportan un servicio o solución dentro de un sistema distribuido, en el campo de funciones de red es aún más complejo debido a la arquitectura necesaria para desplegar Funciones de Red en un ambiente virtualizado. Una vista detallada de los Servicios de Red Orquestados se pueden ver en la figura 3-9.

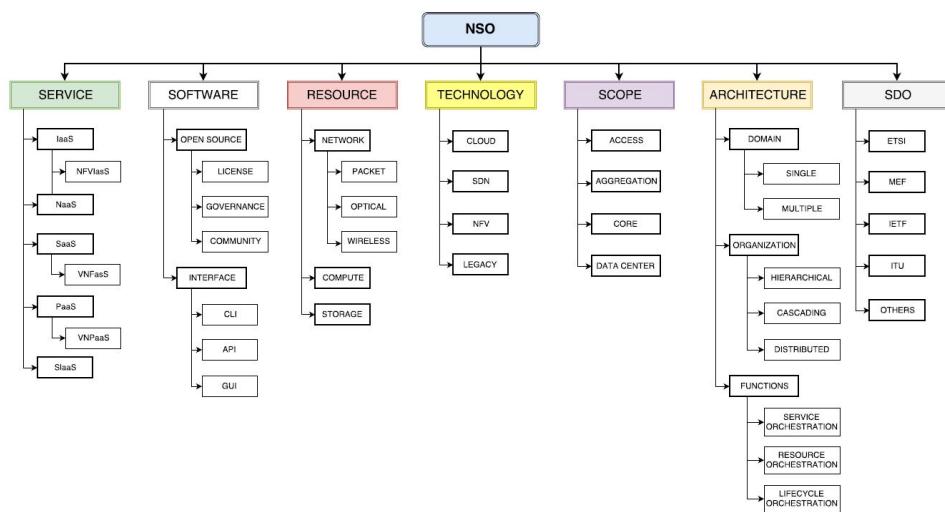


Figura 3-9: Clasificación de NSO de Sousa y cols. (2018)

Capítulo 4

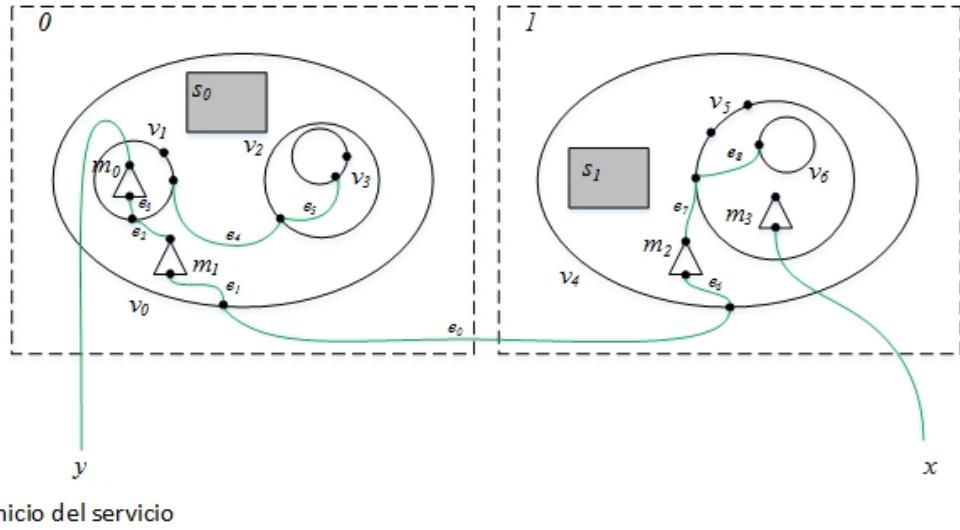
Modelamiento Formal del Sistema

4.1. Modelado con Bigrafos

Dentro de la variedad de álgebras de procesos y cálculos para sistemas distribuidos se ha seleccionando el modelo de bigrafos presentado en el anexo B, para definir las interacciones entre el agente local y el orquestador diseñados para el desarrollo de este trabajo, en este sentido existen dos raíces 0 y 1 o lugares físicos, inicialmente se hace la relación uno a uno, es decir un agente local y un orquestador, pero el modelo puede ser extendido a n miembros del sistema virtualizado inalámbrico construido, el nodo v_0 denota un nodo que contiene un agente local, el nodo v_4 contienen un orquestador v_5 , en el cual v_6 representa la política a implementar en el sistema, en las secciones siguientes se validaran todos los modelos propuestos con múltiples agentes locales y el modelo donde todos tienen el rol de orquestador, el nodo v_1 representa el Agente Local, el nodo v_2 representa el controlador y el nodo v_3 la aplicación, este es el modelo general de control del sistema propuesto, las interfaces están representadas por m_1 y m_2 , mientras los agentes tanto el AL como el OR están presentados por m_0 y m_3 respectivamente, los espacios de almacenamiento dentro del nodo están representados por s_0 en el AL y s_1 en el orquestador. Representando por el bigrafo de la figura 4-1, este bigrafo esta definido por:

$$G : \epsilon \rightarrow \langle 2, \{x, y\} \rangle \quad (4-1)$$

Del mismo modo el grafo de lugar como $G^P : 2 \rightarrow 2$ y el grafo de enlace como $G^L : \{x, y\} \rightarrow 0$

**Figura 4-1:** Bigrafo del sistema Estado 0

En la figura 4-1 se muestra el estado inicial del los nodos en el sistema, aunque la representación esta hecha sobre dos nodos este proceso es realizable con n nodos del tipo v_0 , representados como agentes locales y su respectiva interacción con el nodo v_4 el Orquestador, en esta etapa los nodos aún no están conectados,sin embargo, el código esta en ejecución y existe la interacción entre los elementos nativos del sistema como lo son el agente local del nodo m_0 ,los contenedores de Docker v_2 y las aplicaciones a desplegar v_3 , esto por parte del agente local.

Por el otro lado el orquestador es desplegado v_6 y en espera de las peticiones o la interfaz de usuario (Consola) para enviar las solicitudes (m_3).

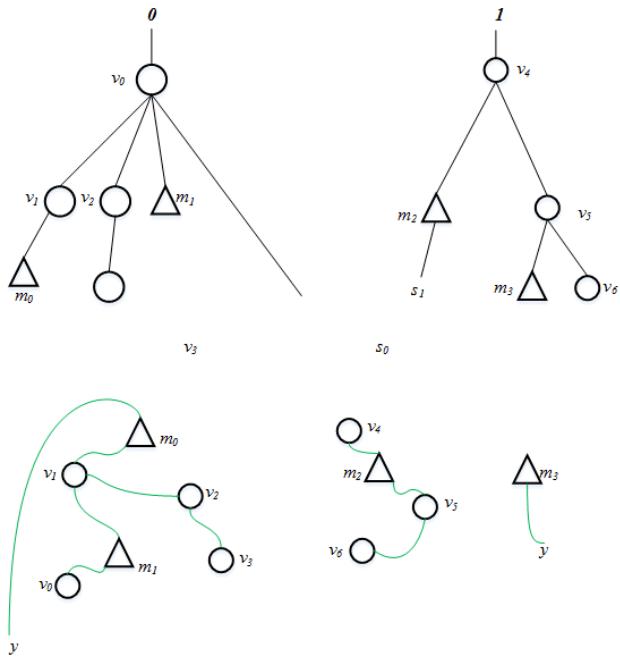


Figura 4-2: Grafo de Lugar e Hipergrafo

Un bigrafo esta compuesto por el grafo de enlace y el grafo de lugar como se ve en la figura 4-2 , el grafo de lugar se puede apreciar en la parte superior donde se muestra las raíces en este caso la ubicación física del nodo y los lugares lógicos de este diagrama es decir los nodos v_x , los agentes m_x y el espacio del almacenamiento s_x , el de la izquierda representa al AL y el de la derecha al OR.

Por su parte el grafo de enlace parte inferior de la gráfica 4-2 representa los enlaces de los nodos y agentes con los nombres de enlaces tanto internos como externos (x, y), este grafo permite identificar las reglas de reacción y los resultados de las interacciones entre los artefactos computacionales y las interfaces físicas de los dispositivos. Es importante indicar que las interacciones de este modelo esta representadas por reglas de interacción, es decir lo cambios de estado del sistema, dados por mensajes e interacciones específicas entre los nodos y específicamente entre el agente local y el orquestador.

Definición del Bigrafo de este trabajo

$$\mathcal{K} = \{\mathcal{N} : 1, \mathcal{A} : 3, \mathcal{I} : 2, \mathcal{C} : 2\} \quad (4-2)$$

$$V = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, m_0, m_1, m_2, m_3\} \quad (4-3)$$

$$E = \{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\} \quad (4-4)$$

$$crtl(v) = \begin{cases} N : 1, & \text{if } v \in \{v_0, v_1\} \\ A : 3, & \text{if } v \in \{v_1, v_4\} \\ I : 2, & \text{if } v \in \{m_0, m_1, m_2, m_3\} \\ C : 2, & \text{if } v \in \{v_3, v_6\} \end{cases} \quad (4-5)$$

$$prnt(v) = \begin{cases} v_0, & \text{if } v \in \{s_0, m_0\} \\ v_1, & \text{if } v \in \{m_1\} \\ v_2, & \text{if } v \in \{v_3\} \\ v_4, & \text{if } v \in \{s_1, m_2\} \\ v_5, & \text{if } v \in \{v_7, m_3\} \end{cases} \quad (4-6)$$

$$link(l) = \begin{cases} e_0, & \text{if } l \in \{v_0, v_1\} \\ e_1, & \text{if } l \in \{v_0, v_1\} \\ e_2, & \text{if } l \in \{v_0, v_1\} \\ e_3, & \text{if } l \in \{v_0, v_1\} \\ e_4, & \text{if } l \in \{v_0, v_1\} \\ e_5, & \text{if } l \in \{v_0, v_1\} \\ e_6, & \text{if } l \in \{v_0, v_1\} \\ e_7, & \text{if } l \in \{v_0, v_1\} \\ e_8, & \text{if } l \in \{v_0, v_1\} \\ x, & \text{if } l \in \{v_1, v_4\} \\ y, & \text{if } l \in \{v_1, v_4\} \end{cases} \quad (4-7)$$

$$P = \{(v_0, 0), (v_1, 0), (v_1, 1), (v_1, 2), (v_2, 0), (v_3, 0), (v_3, 1), \\ (v_4, 0), (v_5, 0), (v_5, 1), (v_5, 3), (v_6, 0), (m_0, 0), (m_0, 1), \\ (m_1, 0), (m_1, 1), (m_2, 0), (m_2, 1), (m_3, 0), (m_3, 1)\} \quad (4-8)$$

Para definir correctamente este bigrafo es necesario indicar que el modelo propuesto esta desarrollado sobre el concepto de bigrafo concreto definido como $b = (V, E, ctrl, prnt, link)$ (ver anexo B), la ecuación 4-2, están indicados los componentes fundamentales del sistema en el Nodo \mathcal{N} con un puerto al igual que las políticas representadas en el orquestador como v_6 , los agentes \mathcal{A} tanto el agente local como el orquestador con tres puertos asociados, las interfaces y el programa de control del agente Local y el orquestador están descritos como \mathcal{I} , las aplicaciones por su parte con dos puertos están representadas como \mathcal{C} , el conjunto de nodos del bigrafo esta descrito en la ecuación 4-3, el conjunto de enlaces del bigrafo esta en la ecuación 4-4, el mapa de control del bigrafo en la ecuación 4-5, el mapa de los padres en la ecuación 4-6 y el mapa de enlaces esta representado en la ecuación 4-7.

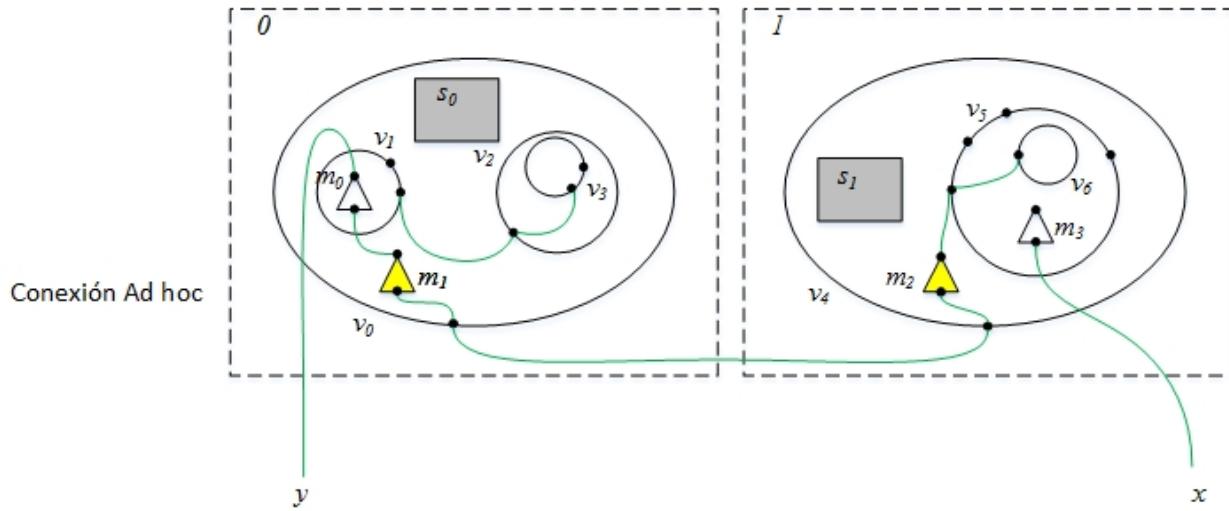


Figura 4-3: Bigrafo Estado 1

Continuando con los cambios de estado del sistema e identificando las interacciones entre el agente local y el orquestador el siguiente estado (figura 4-3) , las interfaces del AL y el OR están enlazadas se activa la interfaz wifi en modo ad hoc, esto permite a los nodos operar en el modo mesh del protocolo IEEE 802.11, en esta etapa se esta formando la red ad hoc y se realiza el mapeo de recursos en el sistema.

En el estado 2 (figura 4-4) se realiza la comunicación entre el AL y el OR por el puerto de identificación, esta es la representación del servicio de descubrimiento de nodos realizado por el orquestador, que permite a su vez definir los roles de cada uno de los nodos miembros del sistema.

En el estado 3 (figura 4-5) se genera una solicitud aplicaciones m_3 y se ejecuta la política v_6 seleccionada por el orquestador, la cual es ejecutada por el nodo seleccionado y asignada mediante el puerto de ejecución del AL, este proceso es una de las interacciones claves del sistema, representa el balan-

Envío de estado
Locales al orquestador

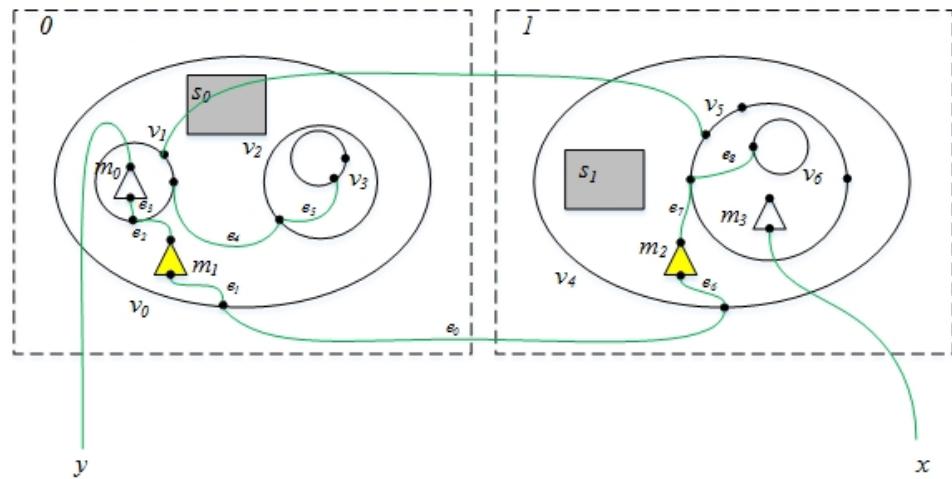


Figura 4-4: Bigrafo Estado 2

Solicitud del
orquestrador

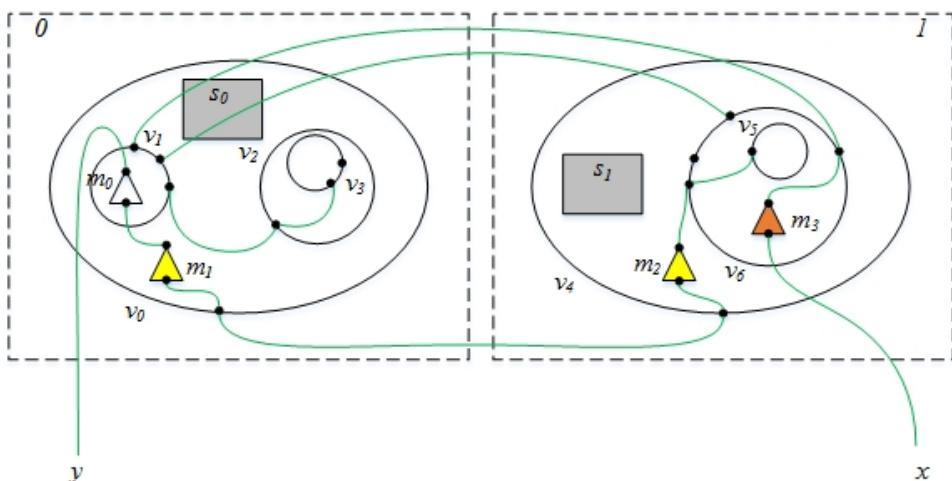


Figura 4-5: Bigrafo Estado 3

ceo de carga realizado por el orquestador sobre el sistema.

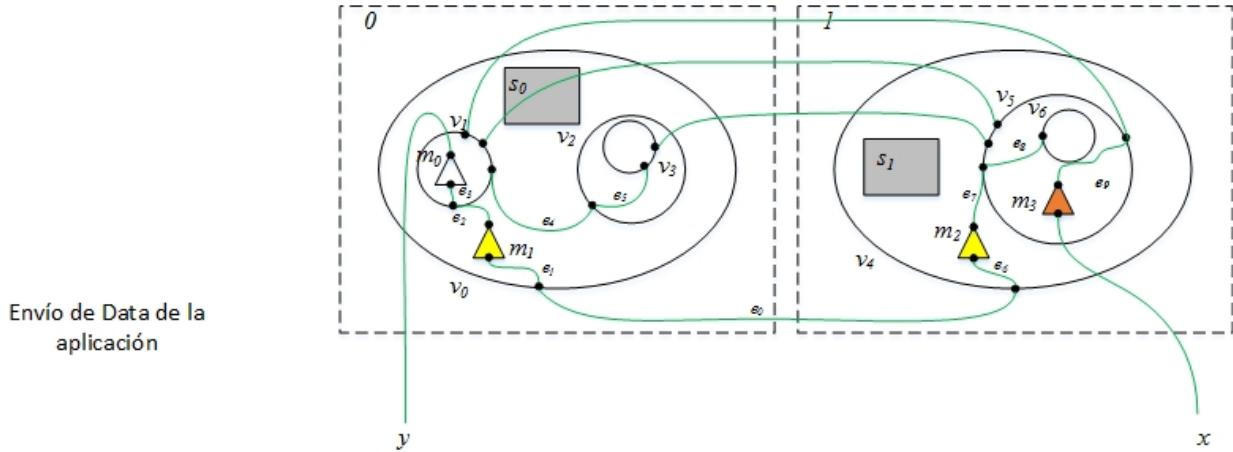


Figura 4-6: Bigrafo Estado 4

El mapeo de recursos y estado del sistema, es alimentado ahora también con la información de la aplicación en ejecución , esta interacción se da entre el contenedor v_2 quien despliega la aplicación v_3 y envía la data generada al puerto de resultados del Orquestador, es importante resaltar que en este trabajo se uso un contenedor por núcleo de cada dispositivo, este proceso se puede ver en la figura 4-6, luego de la ejecución se retorna al estado 1 y se mantiene el proceso, de no existir solicitud del usuario para ejecutar alguna aplicación se mantiene el sistema en el estado 2.

4.2. Reglas de Reacción

Como resultado de estas interacciones se generaron las siguientes reglas de reacción que describen los estados del sistema, el estado del agente Local y del Orquestador, este es uno de los aportes de este trabajo.

Tabla 4-1: Reglas de Reacción

Regla de Reacción	Descripción	Figura
1 R_{idle}	El agente Local y el Orquestador están en ejecución con sin estar enlazados	4-1
2 R_{com}	Las interfaces del AL y el OR están enlazadas se activa la interfaz wifi en modo ad hoc	4-3
3 R_{disc}	Se realiza la comunicación entre el AL y el OR por el puerto de identificación	4-4
4 R_{exec}	Se genera una solicitud aplicaciones m_3 y se ejecuta la política a ser aplicada por el puerto de ejecución del AL	4-5
5 R_{capp}	El contenedor v_2 despliega la aplicación v_3 y envía la data generada al puerto de resultados del Orquestador	4-6
6 R_{mon}	Se retorna al Estado 1 e inicia el ciclo de monitoreo esencial del sistema	4-3

Tabla 4-2: Forma Algebraica de las Reglas de Reacción

Regla de Reacción	Forma Algebraica
$Ridle \rightarrow R_{com}$	$/y(v_0 \mid m_1 \parallel v_1(m_0) \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_2 \parallel (m_3 \mid v_6)s_1 \rightarrow /y(v_0 \mid m_{1e} \parallel v_1(m_0) \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_2e \parallel (m_3 \mid v_6)s_1$
$R_{com} \rightarrow R_{dis}$	$/y(v_0 \mid m_{1e} \parallel v_1(m_0) \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_{2e} \parallel (m_3 \mid v_6)s_1 \rightarrow /y(v_0 \mid m_{1e} \parallel v_1(m_0) \parallel v_6 \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_{2e} \parallel (m_3 \mid v_6)s_1$
$R_{com} \rightarrow R_{exec}$	$/y(v_0 \mid m_{1e} \parallel v_1(m_0) \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_{2e} \parallel (m_3 \mid v_6)s_1 \rightarrow /y(v_0 \mid m_{1e} \parallel v_1(m_0) \parallel v_6 \cdot v_5 \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_{2e} \parallel (m_3e \mid v_6)s_1$
$R_{exec} \rightarrow R_{capp}$	$/y(v_0 \mid m_1 \parallel v_1(m_0) \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_2 \parallel (m_3e \mid v_6)s_1 \rightarrow /y(v_0 \mid m_{1e} \parallel v_1(m_0) \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_{2e} \parallel (m_3e \mid v_6)s_1$
$R_{capp} \rightarrow R_{com}$	$/y(v_0 \mid m_{1e} \parallel v_1(m_0) \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_{2e} \parallel (m_3e \mid v_6)s_1 \rightarrow /y(v_0 \mid m_{1e} \parallel v_1(m_0) \parallel (v_2 \mid v_3) \parallel v_5 \mid s_0) \parallel /x(v_4(m_{2e} \parallel (m_3e \mid v_6)s_1$
$R_{com} \rightarrow R_{mon}$	$/y(v_0 \mid m_{1e} \parallel v_1(m_0) \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_2 \parallel (m_3e \mid v_6)s_1 \rightarrow /y(v_0 \mid m_1 \parallel v_1(m_0) \parallel (v_2 \mid v_3) \mid s_0) \parallel /x(v_4(m_2 \parallel (m_3 \mid v_6)s_1$

Capítulo 5

Computación Social

5.1. Estado del Arte

Los sistemas de cómputo siempre están en constante simbiosis Licklider (1960) con los humanos, en cierto sentido se pueden ver como un modelo colaborativo donde las máquinas proporcionan sus recursos para el desarrollo de alguna tarea específica y en cambio el hombre proporciona los elementos necesarios para el óptimo funcionamiento de las máquinas, en este sentido podemos apreciar una primera relación entre el hombre y los computadores, o tal vez considerar la máquina a la par del hombre.

Para tener una verdadera simbiosis uno de los problemas a resolver es el uso del lenguaje, esta barrera impuesta por el funcionamiento de las máquinas y el pensamiento de los hombres, genera restricciones para una posible simbiosis, esta búsqueda de relaciones o el deseo de que las máquinas imiten a los hombres o poder vislumbrar la existencia de alguna inteligencia en las mismas Turing (2009) ha generado, diversas ramas de investigación en las ciencias de la computación, como la Inteligencia Artificial, Vida Artificial , Sistemas Artificiales, Robotica entre otras.

Bajo estos preceptos se dan varias opciones para llegar a modelos social inspirados, pero en primera instancia es necesario identificar que nos permite ser sociales, inicialmente la existencia de comunicación entre individuos da elementos de sociedad o de organización, bajo este modelo podemos hablar de sistemas Bio inspirados, los cuales tienen la característica de cooperación, o en algunos casos como un comportamiento emergente deseado Nitschke (2005), este tema es contrastado con investigaciones sobre inteligencia artificial distribuida, o la creación de agentes simples con mecanismos que permiten modelar estos comportamientos, ya sea como agentes motores, cognitivos, perceptuales, motivacionales o porque no social inspirado. Un modelo altamente estudiado son los enjambres, ya sean insectos, aves o cardúmenes de peces, esto con el fin de simular el comportamiento del individuo y llegar a generar los comportamientos emergentes de cooperación cuando son miembros de sus respectivas colonias, esto se da gracias a las interacciones entre los agentes y los productos obtenidos

de las mismas.

Pasando a una abstracción mayor se pueden encontrar dos tipos de investigación sobre inteligencia artificial, la primera dedicada a resolver problemas mediante analogías con la física, biología, psicología, entre otras, y la segunda por medio de un enfoque orientado a la construcción de una herramienta metodológica que permita identificar como funciona la mente humana Watt (1997), en psicología hay un creciente interés en la investigación sobre el funcionamiento de la mente, con modelos base como el de deseos, intenciones y creencias, donde recae toda la responsabilidad en el diseño del agente, una palabra sencilla que alberga una gran variedad de significados y manifestaciones complejas en el campo de la computación.

Virando al concepto de agente, este puede ser tan sencillo como un elemento que recibe entradas (sensor), las cuales lo llevan a ejecutar acciones (actuador) sobre un ambiente, o en un contexto más complejo la interacción entre el agente y una persona sobre un ambiente físico y/o social, operando bajo un conjunto de reglas que definen el espacio y limitan el elenco de acciones posibles a realizar y por ende el conjunto de estados Moore y Roger (2016). Un ejemplo de ello es el uso de agentes para definir el desarrollo cognitivo y definir las estrategias de aprendizaje en un estudiante Jung y cols. (2002), de acuerdo a las comunicación entre agentes en diversos niveles, y finalmente el agente control de instructor con el estudiante.

Estas interacciones entre agentes dan cabida a la aparición de algoritmos sociales que tratan de imitar las relaciones humanas en términos de liderazgo, cultura, asociación o coalición, incluyendo las dinámicas sociales humanas como estrategias de optimización Neme y Hernández (2009), este tipo de algoritmos tiene como base considerar el poder de cómputo correlacionado con la riqueza o complejidad de un comportamiento social dado, este comportamiento social es el resultado de una interacción entre individuos, este comportamiento que emerge o resultante no es lineal ni predecible en la mayoría de los casos, no requiere las mismas condiciones para manifestarse en un ambiente particular.

Debido a las interacciones entre las ciencias de la computación, la psicología, la biología, una clasificación más completa de este tipo de algoritmos es propuesta en Kumar y cols. (2018), donde podemos observar las siguientes ramas o líneas de algoritmos socio culturales como se ve en la figura 5-1, una rama esta compuesta por los algoritmos basados en ideologías socio políticas: entre los cuales están los algoritmos de ideología, de elección y de Campaña Elección, otra rama es los algoritmos basados en el comportamiento de competencia deportiva: entre ellos destacan el algoritmo de liga de fútbol y el algoritmo de campeonato, en tercer lugar están los algoritmos basados en interacción cultural , con una gama más variada entre los que están : los algoritmos de enseñanza aprendizaje, optimización de grupo social, aprendizaje social, optimización de emociones y un modelo de algoritmos socio

evolutivos y de aprendizaje Srinivasan y Ramakrishnan (2012), estos últimos con similaridad a los algoritmos genéticos y el cruce de descendencias, finalmente la cuarta rama contiene los algoritmos basados en colonización: esta clasificación hace referencia a los algoritmos de sociedad y civilización, competencia imperialista, y sociedad anárquica.

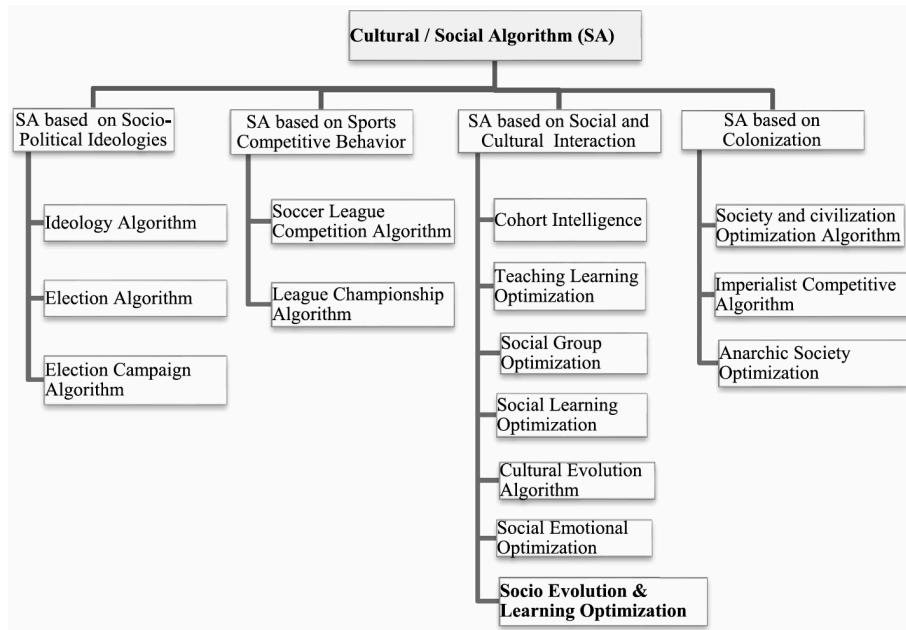


Fig. 2. Broad classification of socio-inspired algorithms.

Figura 5-1: Algoritmos Socio Culturales Kumar y cols. (2018)

¿Es posible modelar los valores y sentimientos humanos sobre un agente o un sistema de inteligencia distribuida?, algunos desarrollos sobre estas temáticas están basados en el concepto de justicia y en algunos postulados filosóficos como los de Jhon Rawls o económicos como los de Elinor Ostrom, estos modelos se pueden evidenciar en una línea de investigación denominada justicia computacional Fitoussi y Tennenholz (2000); Pitt y cols. (2015), al igual que los modelos de institucionalidad de un estado dentro de un sistema multi-agente o un sistema de inteligencia distribuida, estos modelos ideales pueden ser simulados en ambientes controlados y obtener resultados adecuados con técnicas de optimización y la elección y sistematización adecuada de las reglas dentro del ambiente. Finalmente las Tecnologías de la información y las comunicaciones (ICT en inglés), pueden incluir elementos social inspirados como por ejemplo la confianza, normas morales y el contenido ético Ferscha y cols. (2012) como se ve en la figura 5-2, estas características son incluidas en el diseño de agentes y en la validación de comportamientos emergentes, o mejor, clasificados como comportamientos emergentes dentro de un sistema o un conjunto de interacciones.

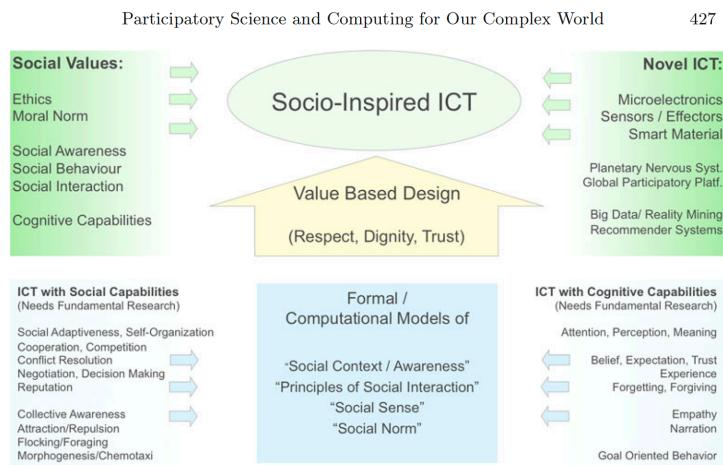


Figura 5-2: Socio Inspiración en ICT Ferscha y cols. (2012)

Estos modelos vistos a groso modo indican una evolución constante en la búsqueda de comportamientos humanos en las máquinas o en el deseo de incorporarlos a los dispositivos de cómputo para resolver problemas o modelar interacciones de formas novedosas, del mismo modo se propone la clasificación de computación social inspirada y bio inspirada basados en el modelo POE (Filogenia, Ontogenia y Endogenia) y el Modelo sarterano del ser (Sartre (1946)) el cual se ve en la figura 5-3 .

5.1.1. Modelo Estado -PseudoEstado

En el desarrollo de este trabajo uno de los retos es realizar la abstracción de las interacciones humanas en sistemas artificiales, para ellos se debe entender el comportamiento de las máquinas, las bases formales de su operación y por supuesto indicar los límites de la misma o la incapacidad de su desarrollo, es probable que el concepto de la maquina de Turing no sea suficiente para formalizar las interacciones de los dispositivos, agentes, usuarios y servicios desplegados en un sistema, incluso el profesor Robert Milner (2009) propone para modelar un sistema UbiCuo, el uso de una Máquina Abstracta UbiCua (UAM siglas en inglés) escalando los modelos computacionales en términos de ubicación, conectividad, movilidad y comportamiento estocástico.

Basados en esta primitiva de interacción y para desarrollar este modelo es importante tener en cuenta la definición de Inmanencia Spinoza (s.f.), de Estado Hobbes (2001) estos con el fin de poderse sumergir en un cambio de paradigma Kuhn (2012).

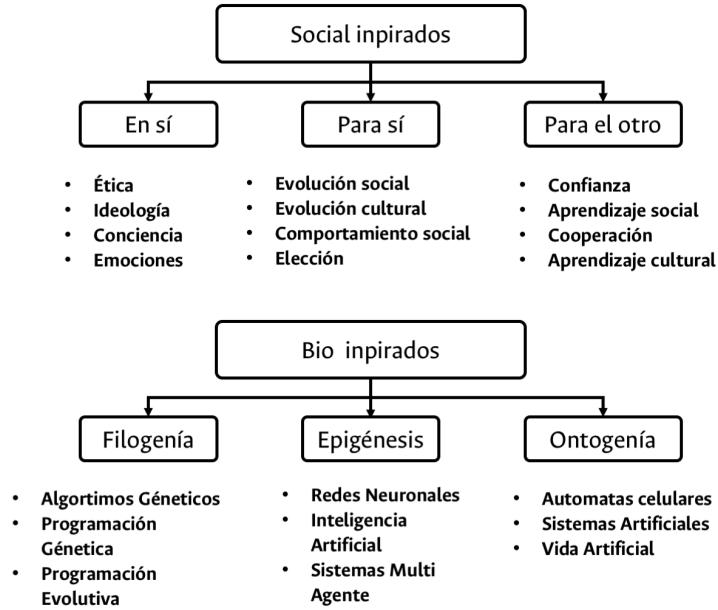


Figura 5-3: Taxonomía Social Inspiración Bio inspiración

5.1.2. Inmanencia

Inmanencia

Toda aquella actividad o comportamiento inherente al sistema de referencia cuya acción ocurre y perdura en su interior, y tiene su razón de ser en ese mismo sistema y que, además, no es el resultado de una acción exterior a los elementos que la componen, aunque su estudio puede realizarse de forma aislada o independiente del sistema. Spinoza (s.f.).

Una manera de formalizar la inmanencia es mediante el operador unión, acorde con las características y el modelo de capas del sistema TLÖN el cual se puede ver en el capítulo 6, esta operación esta presentada en la ecuación 5-1, el conjunto de capas esta representado como 5-2

$$T \equiv \bigcup_{i=1}^n C_i \quad (5-1)$$

donde:

$$C = \{a_j\}_{j=1}^{j=m} \quad (5-2)$$

5.1.3. Justicia

Justicia Distributiva

Se puede definir como la primera virtud de las instituciones sociales. La justicia tomada como imparcialidad indica una distribución de recursos acorde con las capacidades naturales de cada individuo por lo cual todos los miembros de una sociedad, a pesar de su cantidad de recursos, perciben la misma posición social o estatus.

Los elementos que fortalecen el modelo de Rawls son los **Principio de Igualdad** y el **Principio de Diferencia**, el primero enuncia:

“Cada persona tiene el derecho irrevocable a un esquema plenamente adecuado de libertades básicas iguales que sea compatible con un esquema similar de libertades para todos”.

El segundo complementa:

“Las desigualdades sociales y económicas tienen que satisfacer dos condiciones: en primer lugar, tienen que estar vinculadas a cargos y posiciones abiertos a todos en condiciones de igualdad equitativa de oportunidades; y, en segundo lugar, las desigualdades deben redundar en un mayor beneficio de los miembros menos aventajados de la sociedad”.

Bajo estos preceptos ha sido desarrollados modelos de justicia computacional distributiva que no son implementados en este trabajo pero forman parte del diseño del Sistema TLÖN.

5.1.4. Estado

El concepto de estado ha sido tratado desde los inicios de las civilizaciones, en nuestro caso la definición más concreta es la de Thomas Hobbes "Dícese que un Estado ha sido instituido cuando una multitud de hombres convienen y pactan, cada uno con cada uno, que a un cierto hombre o asamblea de hombres se le otorgará, por mayoría, el derecho de representar a la persona de todos (es decir su representante)."

La esencia del Estado

Una persona de cuyos actos una gran multitud, por pactos mutuos, realizados entre sí, ha sido instituida por cada uno como autor, al objeto de que pueda utilizar la fortaleza y medios de todos, como lo juzgue oportuno, para asegurar la paz y defensa común

LA figura 5-4 tiene como objeto presentar la analogía del modelo de pseudo estado con base en los conceptos de Estado y Justicia mencionados en esta sección, como elementos base están las leyes naturales tales como justicia, equidad, modestia, piedad y, en suma la de haz a otros lo que quieras a ti

Hobbes (2001), estas son leyes base del estado quien imparte justicia a sus ciudadanos con la creación de leyes civiles que mantengan la paz y el bienestar de la población. En este sentido la propuesta de pseudo estado es generar pseudo políticas (abstracciones computacionales), que permitan mantener un uso adecuado de los recursos físicos y virtuales distribuidos en una red ad hoc, (territorio del pseudo estado), donde son ejecutados por agentes (ciudadanos) de este modelo social, en primera instancia estos agentes o medios son quienes operan las instituciones que permiten la operación del pseudo estado, es decir, los artefactos computacionales que velan por la operación del sistema de cómputo distribuido TLÖN, este modelo y sus interacciones se ve en la figura 5-4

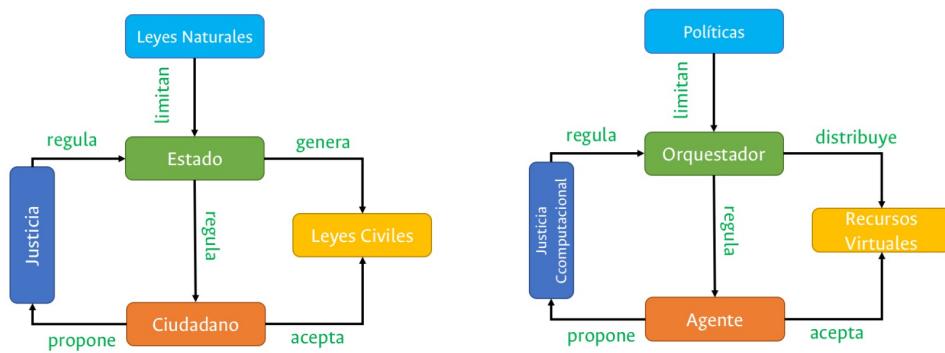


Figura 5-4: Modelo de Estado. Adaptado de Hobbes (2001)

5.1.5. Modelos de Pseudo Estado Propuestos

Antes de iniciar con un modelo particular de pseudo estado es necesario introducir los tipos de estado y las formas de gobierno, esto nos permite dimensionar las opciones dentro del modelo planteado en este trabajo, al igual que generar categorías de operación dentro de una red ad hoc, es importante indicar que existen varios estudios sobre los modelos de estado y dependen enteramente de la época y de la evolución social de la humanidad, incluso varía radicalmente de la ubicación y madurez de las sociedades. Dos elementos inseparables son la forma de Gobierno y la forma de Estado, como una representación bidireccional podríamos decir que:

Manteniendo esta relación podemos distinguir tres formas de estado: Unitario, Regional y Federado, las cuales dependen específicamente de la cantidad de personas que tienen la potestad de tomar decisiones y generar las leyes o reglas del caso. Según Aristóteles se pueden clasificar los estados dependiendo la cantidad y la representación positiva o negativa por parte del monarca o Leviatan si lo tomamos en el lenguaje de Hobbes, en la tabla 5-1 se muestran las formas de estado en detalle.

Quien	Forma Positiva	Forma Negativa
Uno	Monarquía	Tiranía
Pocos	Aristocracia	Oligarquía
Muchos	Democracia	Anarquía

Tabla 5-1: Tipos de Estado

En cuanto a las formas de gobierno el panorama no es muy diferente, podemos identificar dos clasificaciones la República y la Monarquía, de esta manera se dan dos casos la elección o la asignación por herencia del gobierno como se muestra en la tabla 5-2.

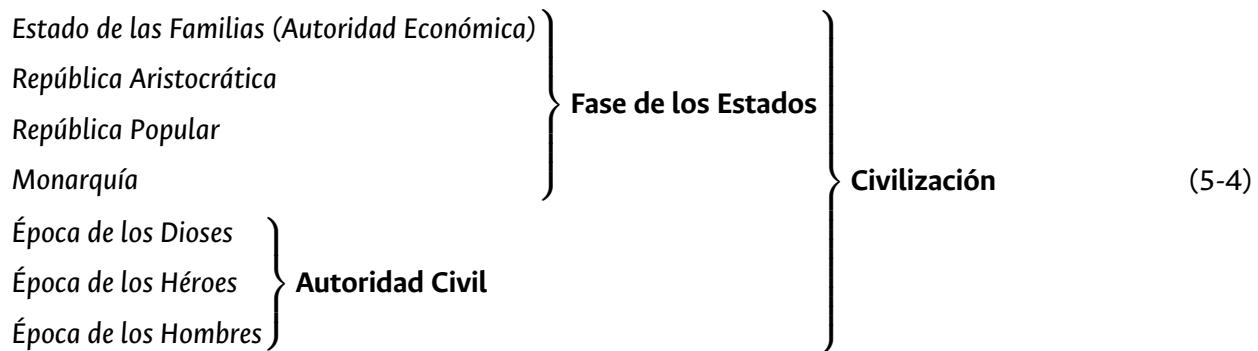
República	Monarquía
Parlamentaria	Parlamentaria
Presidencialista	Semi constitucionalista
Unipartidista	Absoluta

Tabla 5-2: Formas de Gobierno

En este contexto Hobbes en analogía con el cuerpo humano indica que existen tres facultades que deben ser administradas e irrigadas al aparato del estado y en consecuencia a sus miembros, la facultad nutritiva representada por el poder financiero (Dinero), la facultad locomotriz del estado (La conducta) y la facultad racional (Las leyes). Estas facultades son las propuestas a ser usadas en el presente proyecto, en particular la facultad racional y locomotriz, la facultad nutritiva esta distribuida dentro del sistema TLÖN en el Sistema Multi agente.

Una reseña corta y muy completa es propuesta por Giambattista Vico en su obra *ciencia nueva* Vico y cols. (1995), donde muestra una evolución de los estados a través de la historia y los acontecimientos

históricos, muy acorde a las dinámicas humanas actuales como se ve en 5-4.



5.1.6. Leyes Naturales y Leyes Positivas

Para Thomas Hobbes las leyes permiten al soberano, gobernar pero del mismo modo debe ser el primero en respetar las leyes para mantener la paz y el bien común, para Hobbes existen dos tipos de leyes las *naturales* y las *positivas*. Las *leyes naturales* son aquellas que han sido leyes por toda la eternidad y no solo se llaman leyes naturales también llamadas *leyes morales*, porque descansa en las virtudes morales como la justicia, la equidad, la caridad y en general los hábitos que conducen a la paz. Por su parte las *leyes positivas* son las constituidas como leyes por la voluntad de quienes tuvieron poder soberano sobre otros, y son formuladas, escritas y dadas a conocer por algún instrumento dado por su legislador, por ejemplo una constitución.

En las leyes positivas se pueden clasificar en las leyes *distributivas y penales*, las primeras son las que determinan los derechos a los súbditos, las que son dirigidas a todos los miembros del estado. Son *penales* las que declaran que penalidad se debe infligir a quienes violan la ley, su ejecución se denomina sentencia.

Dentro del modelo Propuesto se postularán leyes Positivas y se indicaran las naturales, estas vitales para el desarrollo de las políticas dentro del sistema y las interacciones de los miembros del sistema, es decir de los artefactos propuestos los cuales se pueden ver en el capítulo 7.

Capítulo 6

Proyecto Sistema TLÖN

El proyecto del Grupo de investigación en redes de Telecomunicaciones Dinámicas y Lenguajes de Programación Distribuidos – TLÖN, propone un esquema de computación inspirado en modelos Sociales, inviables en la práctica pero posibles en entornos artificiales controlados, este sistema basado en los conceptos de Justicia de Jhon Rawls, Inmanencia de Baruch de Spinoza, Paradigma Thomas Kuhn, Estado Thomas Hobbes y las concepciones de existencia y esencia de Jean Paul Sartre, estos elementos deben reflejarse en abstracciones computacionales para su despliegue más básico, el objetivo es encontrar una analogía completa de un esquema de virtualización inalámbrica, necesaria para implementar estos modelos sociales en sistemas computacionales estos principios se ven descritos en la figura 6-1. Este modelo social inspirado, es una abstracción superior a los modelos bio-inspirados, se pueden observar los componentes del modelo propuesto, el cual además funcionará sobre una red Ad Hoc con todas sus condiciones dinámicas, estocásticas e inalámbricas.

Dentro del esquema general de la propuesta se incorpora un elemento capaz de integrar los sistemas multiagente con la virtualización, con el comportamiento de una red Ad Hoc y las necesidades de servicios, por ejemplo en situaciones de emergencia. Este elemento contenedor del sistema es un Lenguaje de Programación, que básicamente lo que hace es tomar un lenguaje de alto nivel y convertirlo en un lenguaje entendible para las máquinas, es decir un lenguaje de bajo nivel, algunos prototipos básicos de lenguajes para agentes han sido realizados pero con limitaciones claras en rendimiento o acciones que no requieren un análisis ni un componente computacional alto.

La mejor forma de mostrar las dimensiones del proyecto TLÖN es por medio de un modelo de capas como se ve en la figura 6-2, en la primera esta la infraestructura, donde se encuentra la red ad hoc, la segunda capa contiene la virtualización inalámbrica y el Sistema Operativo, en tercer lugar se cuenta con el sistema Multiagente, donde operaran comunidades de agentes que proveerán servicios a lo largo de la red ad hoc en el esquema de virtualización y finalmente una capa de aplicaciones específicas del sistema de cómputo. De forma transversal se tiene el lenguaje del sistema y la ontología propia

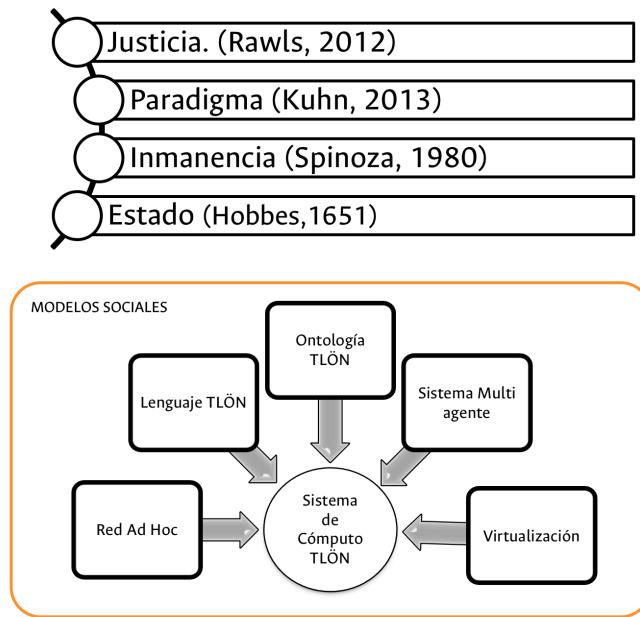


Figura 6-1: Principios de la Social inspiración.

del mismo, generando las interacciones en todas las capas del modelo.

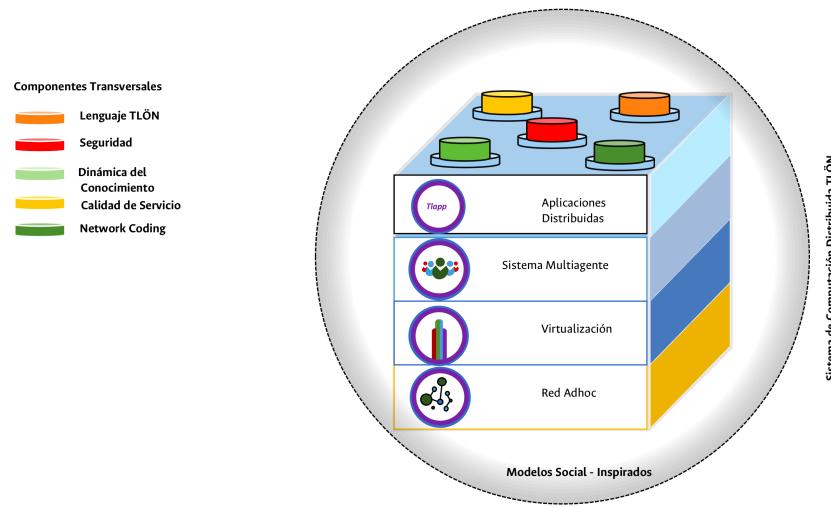


Figura 6-2: Sistema TLÖN .

En la primera fase de la computación gobernaban los sistemas monolíticos, hoy en día algunas aplicaciones y sistemas siguen este modelo, pero las necesidades de comunicación, las velocidades de transmisión y los servicios proliferados a través del mundo, obligaron a ver las aplicaciones y los servi-

cios, como sistemas distribuidos, estos modelos permiten la interacción entre los diferentes equipos, aplicaciones y la distribución de tareas entre varias estaciones de cómputo, básicamente es el proceso de asignar tareas a los miembros de un sistema para mejorar la prestación de uno o varios servicios, al igual que los tiempos de respuesta , que en ultimas influyen en la percepción del usuario sobre el servicio.

Es en este contexto y con la evolución constante de los dispositivos de cómputo(Hardware), la miniaturización y la rápida evolución de sistemas inalámbricos, aparecen sistemas que deben romper con los esquemas tradicionales de computación, siendo este un momento para integrar diversos campos de la computación en una sola solución. Los elementos claves dentro de esta solución son un **Lenguaje de Programación**, el cual permite al usuario o programador adaptar la solución a diferentes contextos, donde diversos servicios puedan ser desplegados, teniendo en cuenta los recursos de cómputo disponibles, otro elemento clave es la **Abstracción Lógica** que permite integrar estos recursos distribuidos, en este caso hardware, como un elemento lógico (software) para la distribución de tareas, esta abstracción es materializada mediante un artefacto conocido como virtualización, en este caso de recursos, servicios y sistemas operativos, este despliegue permite monitorear y generar esquemas de control dentro del sistema TLÖN.

Estos sistemas comunes en aplicaciones de computación en la nube, Data Center o el mismo Internet, tienen un elemento en común, el procesamiento de grandes volúmenes de datos, este crecimiento de información va en contra de la capacidad y cantidad de recursos computacionales contenidos dentro de nuestro sistema "solución", o simplemente la respuesta a estas peticiones es muy lenta, la complejidad computacional aparece y campos como la algoritmia son necesarios para solucionar en un tiempo razonable las operaciones requeridas para desplegar un servicio o un sistema, por ello es necesario tener en cuenta técnicas de análisis de datos, procesamiento de información y predicción de sucesos, es allí cuando aparece otro nivel clave dentro del sistema "solución", **la inteligencia artificial**, este nivel dota a la solución de capacidades de inferencia y procesamiento a partir de la figura del agente, conocido simplemente como un script, que recibe o sensa datos, los procesa bajo algún algoritmo de decisión y finalmente actúa, o entrega una salida, ya sea como data o como una acción puntual en el sistema, este elemento, en cierto modo sencillo permite al programador de la solución amplificar la capacidades del sistema y controlar si es necesario, su comportamiento dinámico, estocástico y disruptivo si este se presenta.

Estos elementos o niveles de abstracción, conforman el sistema TLÖN, un sistema de cómputo distribuido que permite interactuar como un solo elemento conformado por diversos nodos o elementos, con la gran particularidad de estar inmersos en una red inalámbrica, móvil, auto configurable y con la necesidad de adaptarse a las condiciones adversas en recursos, conectividad y existencia. Este tipo de redes conocidas como redes ad hoc, no son la única estructura donde este sistema puede desplegarse, son estas condiciones las que lo ponen a prueba y se asemejan a las condiciones actuales de los usuau-

rios, sistemas móviles, dinámicos, con acceso a Internet y a un sin fin de aplicaciones, es el ingreso al dominio del usuario y de su componente social, lo que impulsa a la aparición de este sistema solución.

¿Cómo desplegar un Sistema de Cómputo Distribuido TLÖN?, como cualquier sistema contiene principios, leyes y restricciones, este sistema denominado social-inspirado, toma como base el concepto de Justicia de Jhon Rawls, justicia distributiva como base para la asignación de recursos entre la abstracción y la percepción de los usuarios finales. La social inspiración como modelo de cómputo propone controlar las interacciones e incluir el componente ético dentro de los sistemas de cómputo, imposible en sociedades reales pero posible en sociedades artificiales, o simplemente comunidades de agentes artificiales o servicios que actúan juntos en busca de entregar una calidad de servicio adecuada y coherente a los miembros de este sistema, es decir los usuarios finales, propietarios de los dispositivos de cómputo, quienes donan sus recursos para componer este Sistema.

Este modelo idealiza los principios de Justicia, Inmanencia y da paso a principios derivados, como la equidad, la solidaridad y a la aparición de virtudes como la verdad, dentro de este Sistema Artificial, las interacciones permitirán la aparición de nuevas entidades como comportamiento emergente e incluso darán paso a nuevos modelos de Estado, estos dependen de la función, o en términos computacionales del elemento a optimizar, porque al fin de cuentas, todo es un juego de optimización, entonces la pregunta es: ¿Qué queremos optimizar? el tipo de configuración de un sistema depende de los objetivos y funciones que se pretenden maximizar, es por ello que el Sistema TLÖN es un sistema altamente adaptativo.

6.1. Formalización de los Principios Sociales

Conociendo los elementos iniciales del sistema, ¿qué se debe hacer con ellos?, cada una de estas piezas ensambladas deben tener uno o varios principios orientadores que garanticen no solo su correcto funcionamiento, sino una percepción del usuario de un grado de servicio y un conjunto de servicios necesarios para la toma de decisiones. Adicionalmente los dispositivos de una red Ad Hoc al igual que el de una red inalámbrica común, son heterogéneos, es decir no hay un nivel de recursos que permita a cada usuario obtener un nivel de servicio adecuado, más aún en una situación de emergencia. Es allí cuando emerge un concepto clave en el desarrollo de esta propuesta la nube móvil la cual es una plataforma flexible que explota recursos distribuidos, recursos que pueden ser conectados de forma inalámbrica, intercambiados, movidos o aumentados, en general combinados en formas novedosas Fitzek y Katz (2013).

La analogía de este sistema se pude observar en la figura 6-3, donde se ve la estructura del estado y la del pseudo estado, de esta forma se da una relación uno a uno de las abstracciones necesarias para despegar un sistema TLÖN en este caso, el territorio dentro de una red ad hoc, las instituciones como

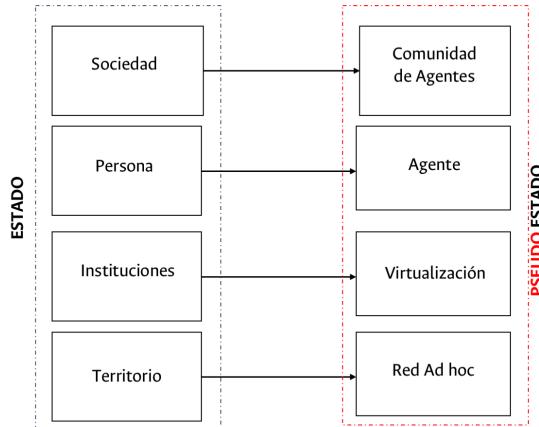


Figura 6-3: Modelo Social Inspirado.

la virtualización, las personas como agentes y la sociedad como comunidades de agentes que prestan servicios específicos.

Este concepto interpretado como una red ad hoc distribuida multi servicios, capaz de ofrecer un grado de servicio igual a cada uno de sus usuarios sin importar las características en recursos de los dispositivos usados, es un modelo social inspirado si se ve desde la perspectiva de los agentes o un modelo aumentado si se ve desde la vista de los recursos. La solución a la vista es un lenguaje de dominio específico que permita integrar estas características en dispositivos con interfaces inalámbricas, formando comunidades de agentes basadas en una sociedad ideal Rawls (2012), redes multi servicio y un lenguaje con un componente nativo. En este caso un esquema de virtualización donde se contengan las reglas de las comunidades de agentes que prestaran los servicios aumentados sobre las interfaces de red y además las características específicas de un lenguaje y un entorno de aplicación que permita la implementación de redes Ad hoc capaces de soportar servicios de telecomunicaciones en diversos campos.

Al igual que la sociedad, las comunidades de agentes deben poseer componentes establecidos para ser regidos y organizados de forma justa, pero ¿El concepto de justicia a qué se refiere? Desde la óptica de los hombres el esquema diverge según los principios de cada individuo, pero en un sistema multiagente la sociedad sería idealizada y puede ser vista desde el velo de ignorancia (Rawls, 2012), por lo cual Rawls afirma, “esto asegura que los resultados del azar natural o de las contingencias de las circunstancias sociales no darán ventajas ni desventajas al escoger los principios”. En una sociedad humana esto es virtualmente imposible de lograr, pero un ambiente parcialmente controlado como un sistema multiagente es totalmente posible, una sociedad ideal compuesta por ciudadanos ideales

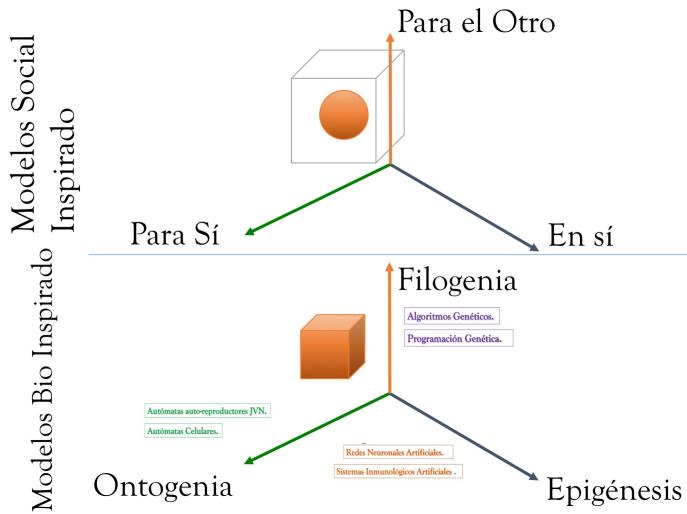


Figura 6-4: Modelo Sartreano.Sartre (1946)

con percepciones y principios diferenciados que permitan obtener un bien común.

Pero llevándolo aun nivel superior el agente debe adoptar el modelo Sartreano para ser desplegado, en un dimensión social, así como en modelos bioinspirados esta el modelo FOA (Filogenia , Ontogenia, Epigénesis), se propone una abstracción mayor que es el modelo de tres dimensiones del ser descrito por Jean Paul Sartre Sartre (1946), donde se deben tener en cuenta las dimensiones del ser como se ve en la figura 6-4, **el en sí**, es decir lo que soy (pasado - presente), **el para sí** lo que deseo ser (futuro) y **el para el otro**, que es el ser que represento para mi prójimo, este componente es el más complejo de realizar en el sistema TLÖN, ya que debe incluir el reconocimiento del otro como par y ser capaz de generar esa percepción de esencia entre los agentes artificiales.

Dentro del contexto de agentes estos se alimentan de percepciones, en nuestro caso serian los requerimientos de los usuarios (servicios), están inmersos en un ambiente: la red ad hoc, están regidos por unas normas, se propone un modelo de pseudoestado, donde reside la capa de virtualización del sistema que entrega las normas a los agentes para asegurar niveles de servicio y rendimiento iguales para todos, estas normas o constitución al igual que un estado son las bases de una nación, para el modelo la nación es la red ad hoc y el ambiente monitoreado por los agentes. Este modelo requiere de un manejo de recursos distribuidos a lo largo de la existencia de la red para poder impartir servicios con justicia, por lo que se generará un mecanismo de dispersión de agentes y modelos de recursos compartidos, ya sea como un super recurso, un intercambio de recursos entre nodos o simplemente recursos aumentados de cada uno de los elementos disponibles en la red ad hoc.

6.2. Modelo Social Inspirado Propuesto - Capa Virtualización inalámbrica

La virtualización es la abstracción lógica que permite administrar los recursos, las interacciones y las comunicaciones entre las capas, dentro del Sistema TLÖN, la unidad mínima de esta capa es el contenedor, el cual aloja los scripts, librerías nativas y elementos necesarios para el despliegue del sistema en su totalidad, un conjunto de contenedores, genera una o varias capas, un elemento de control debe existir dentro de este sistema, el cual se puede ver como una entidad, esta entidad está distribuida a lo largo de los sistemas que componen los clúster de Sistemas TLÖN, es necesario indicar que un sistema TLÖN puede desplegarse incluso en un solo dispositivo, que contenga la cantidad mínima de recursos para operación, un sistema distribuido masivo TLÖN es posible y puede ser considerado como un ecosistema digital similar al del concepto de Internet de las cosas- IoT-, incluso a sistemas como Redes Superpuestas, ciudades inteligentes, todas con grandes cantidades de información.

El esquema de control predictivo es el de Orquestador, este sistema permite la auto adaptación y el control distribuido de las tareas, lo que denomino entidad, es un control distribuido que varía a lo largo del tiempo y el espacio de operación del sistema TLÖN, esta entidad contiene las políticas necesarias para la toma de decisiones a partir de los criterios de entrada, este sistema en conjunto es el sistema operativo Virtual del Sistema TLÖN SOVORA que se presenta a continuación, donde se abstraen y virtualizan los recursos de cada uno de los nodos miembros del Sistema TLÖN.

Capítulo 7

S.O.V.O.R.A.

7.1. Primeras interpretaciones

Para la construcción del sistema distribuido virtualizado inalámbrico, es importante tener una vista modular e identificar los servicios básicos necesarios para su operación, a pesar de tener un esquema formal, o un diseño teórico es necesario limitar el sistema de acuerdo a la arquitectura de los dispositivos actuales y de las capacidades computacionales de los mismos, para tener una referencia de la operación del mismo e identificar las restricciones de la tecnología actual, por lo cual es necesario delimitar las capas del sistema TLÖN, teniendo en cuenta la definición de sistema distribuido, ya sea como un modelo en el cual los componentes localizados sobre una red de computadores comunican y coordinan sus acciones por el paso de mensajes o como el conjunto de computadores que actúan, trabajan y operan como si fueran un gran computador, este es uno de los principales aportes de este trabajo.

De acuerdo con el teorema CAP (Capacity, Availability and Partition Tolerance), postulado por Erick Brewer, referenciados en la sección 2, Consistencia (es la entrega de una respuesta correcta por parte de un servidor a una petición realizada), Disponibilidad (Cada petición eventualmente recibe una respuesta) y Tolerancia a Particiones (son las fluctuaciones de las comunicaciones entre servidores, las cuales pueden ser particionadas en múltiples grupos que no se pueden comunicar unos con otros), se úeden agregar los resultados del modelo FLP (Fischer, Lynch y Patterson) Fischer y cols. (1985), donde se presenta el teorema del consenso para garantizar en un sistema asíncrono la respuesta a una solicitud hecha, se indica que no es posible en un sistema de cómputo distribuido tener los componentes completos del teorema CAP. Por lo cual en el sistema propuesto, se tendrá el modelo de operación bajo AP, disponibilidad y tolerancia a particiones, con el fin de priorizar la conectividad en

la capa física (Red ad hoc) y ser tolerante a particiones, con conceptos como replicación y llegar a niveles de alta disponibilidad.

Como factores determinantes dentro del diseño están los conceptos de orquestador y virtualización ligera vistos en el capítulo 3, claves para desplegar sobre estas abstracciones el sistema operativo virtual, pero del mismo modo es importante definir el modelo de aplicaciones que serán ejecutadas sobre este sistema distribuido inalámbrico, en particular se presentará la arquitectura de microservicios (sección 7.2) la cual nos proporciona características deseadas de este sistema como la partición y recopilación, este modelo de aplicaciones es uno de los aportes de este trabajo.

Todo modelo requiere una formalización y un soporte matemático que permita explotarlo y escalarlo incluso con las limitaciones tecnológicas, de acuerdo con las características de las redes Ad hoc presentadas en el Anexo A, se usa el marco conceptual y teórico presentado en el anexo B, el modelo de Bigrafos proponiendo un modelo formal de las interacciones entre los miembros de la virtualización y las reglas de interacción entre todos los miembros del sistema.

Finalmente el objeto de este proyecto es evidenciar la viabilidad de agregar modelos sociales dentro de sistemas artificiales como lo hacen los modelo biológicos y demás sistemas como los presentados en el capítulo 5, la creación de políticas social inspiradas y los modelos de estado permite la validación de la posibilidad de social inspiración en computación, teniendo en cuenta los principios de Inmanencia, Justicia y el modelo de Estado de Thomas Hobbes.

7.2. Estructura del Sistema y Microservicios

Las arquitecturas de sistemas de cómputo están centradas en modelos monolíticos, que por tradición han sido usados en sistemas donde no hay un número elevado de transacciones, modelos como SOA son efectivos en algunos de estos ambientes, sin embargo en sistemas distribuidos, y más en sistemas estocásticos es necesario validar nuevas arquitecturas para la ejecución de procesos y asignación de recursos para los mismos, en este sentido los microservicios Newman (2015) son servicios pequeños y autónomos que trabajan juntos y pueden ser vistos con la arquitectura mostrada en la figura 7-1.

Una Arquitectura basada en microservicios esta basada en un simple concepto: la creación de un sistema desde la colección de servicios pequeños y aislados, cada uno de ellos con su propia data, y están independientemente aislados, son escalables y resilientes a fallas. La arquitectura de microservicios se basa en una premisa "Haz una sola cosa, pero hazla bien", esta premisa es también usada por los creadores de Linux, donde cada proceso ejecutado es realizado de forma aislada, segura con

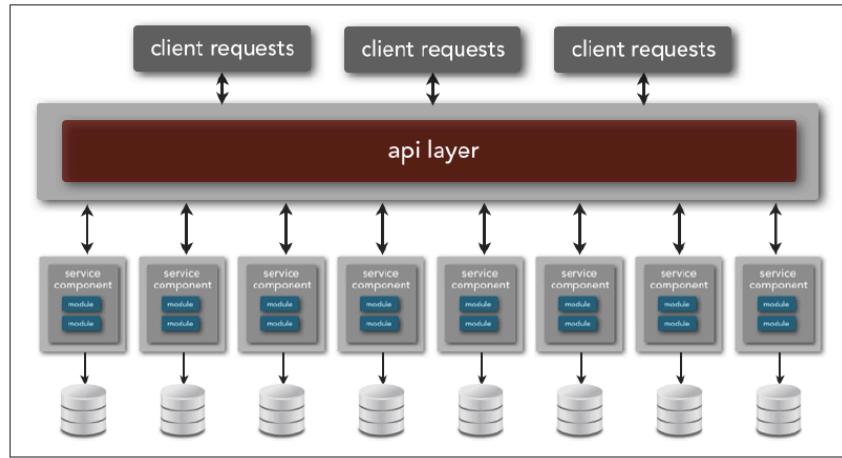


Figura 7-1: Microservicio Newman (2015)

la característica de entregar la información en esquemas compartidos.

Las características claves en una arquitectura basada en microservicios son :

- **Descomposición:** de los sistemas en subsistemas discretos y aislados comunicados sobre protocolos bien definidos.
- **Aislamiento:** es un pre requisito para la resiliencia y la elasticidad, requiere comunicación asíncrona, esto permite desacoplar los límites de los microservicios en:
- **Tiempo:** Permitir la concurrencia.
- **Espacio:** Permitir la distribución – Movilidad, es decir la habilidad de mover servicios alrededor del sistema.

Uno de los requisitos importantes en el diseño de un sistema distribuido es el paso de mensajes y la distribución de la información, el conocimiento de los estados globales y los estados locales de los miembros del sistema es decir un problema de búsqueda de información en bases de datos, file systems o similares, una sola base de datos puede sufrir en tiempos de espera extensos y no procesar las interacciones de los diferentes estados de un proceso, en cambio en el uso de bases de datos individuales por cada unidad de servicio, con un fuerte esquema de comunicación se puede proveer la solución para conocer con precisión el estado de cada unidad o microservicio, cada módulo del sistema operativo distribuido puede verse como un microservicio que se comunica con los diferentes miembros del sistema o módulos replicados, esta característica permite tener propiedades de redundancia

7.3. Modelo de la solución

El modelo propuesto para el sistema operativo SOVORA es un modelo de contenedores, a pesar de estar desplegados sobre el sistema operativo, existe una jerarquía descrita en las capas del sistema TLÖN, (figura 7-2), la primer capa usa el sistema operativo nativo, en este caso la solución funciona sobre kernel de Linux, los primeros módulos son necesario para establecer las estrategias de comunicación , implementar el modo de operación ad hoc de la tarjeta de Red, en este trabajo se usa el módulo BATMAN en el esquema de Comunicación, el servicio de Mensajería en este caso ALFRED es usado como modulo de descubrimiento de nodos, al igual que el motor de docker ce, el modelo de contenedores permite hacer la transición de redes de borde hasta llegar a computación en la nube, como se ha visto en el capítulo 3, del mismo modo la interacción de la solución con lenguajes de programación nativos del Kernel, en este caso se usó Python 2,7 y C++, la segunda capa esta compuesta por tres elementos, el primer es el módulo de comunicación que establece comunicación con el módulo ad hoc del kernel y genera los servicios clientes servidor del sistema, a el mismo nivel esta el balanceador de carga el cual define por medio de la estrategia de control de flujos generar las abstracciones de Plano de control y Plano de Datos, estos módulos son gestionados por el Agente Local (Contenedor), el cual esta presente en cada nodo (mínimo uno por nodo), donde realiza las tareas de monitoreo, ejecución de tareas y solicitud de servicios. Uno de los servicios básicos es el monitoreo del estado local de los recursos y estado de las aplicaciones. Cada una de estas aplicaciones esta soportada en un contenedor y tiene una operación aislada, respecto a los otros artefactos desplegados en el sistema, otro tipo de prueba es distribuir estas porciones de código entre los diferentes nodos, con el artefacto agente móvil el cual permite hacer migraciones entre los miembros del sistema.

El siguiente nivel esta compuesto por el Orquestador el cual gestiona las solicitudes y controla las capas inferiores y superiores a su nivel, esta abstracción permite al usuario hacer solicitudes y visualizar servicios en línea, contiene elementos como la consola para el acceso de los usuarios al sistema, las políticas en este caso basados en los principios de social inspiración vistos en el capítulo 5, el programador de tareas quien distribuye las acciones del sistema dependiendo de los resultados obtenidos al ejecutar las políticas incluidas en el módulo. El módulo del agente móvil permite hacer la migración de un artefacto de software, conservando su estado y su data, es uno de los servicios claves en ambientes mas complejos, finalmente tiene el estado global del sistema, con la información enviada por los nodos, la cual es insumo también de las políticas.

La capa Superior representa el modelo de aplicaciones las cuales tienen la capacidad de dispersión y recopilación, para su ejecución distribuida, un sistema de monitoreo incluido en el contenedor de la aplicación al igual que el balanceador local para realizar las acciones indicadas por el balanceador de la capa 2. Un elemento fundamental en este diseño es el uso de un Sistema Multiagente para poder despegar aplicaciones orientadas a agentes, como parte de este sistema, proporcionando la capacidad de tener inteligencia artificial en los servicios desplegados.

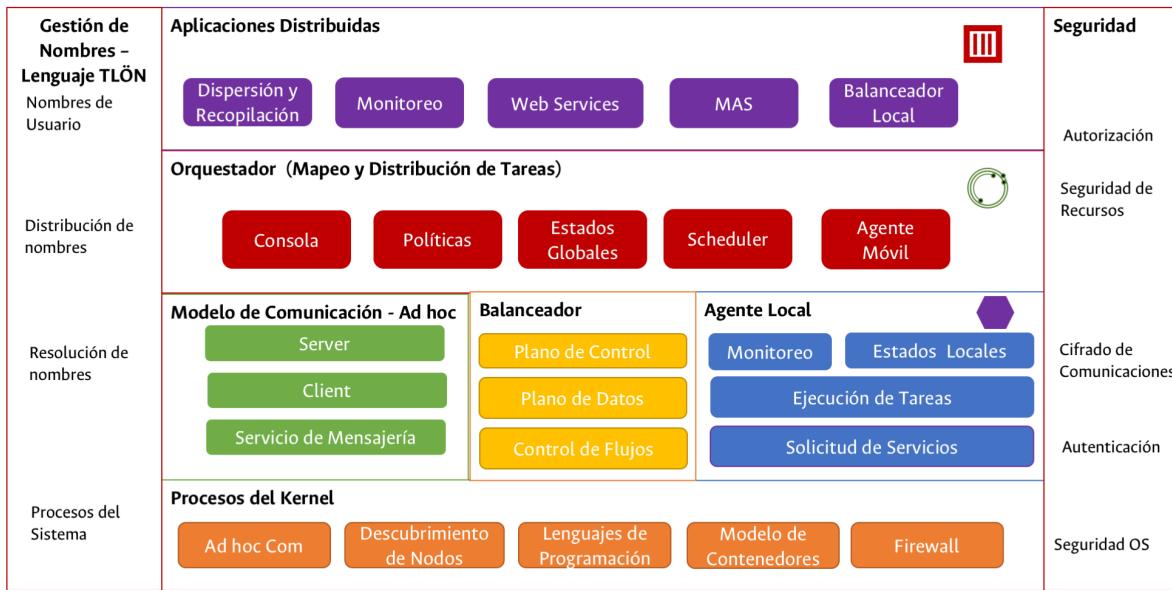


Figura 7-2: Modelo Propuesto para SOVORA

En conclusión este modelo de Sistema operativo basado en agentes locales, orquestadores y contenedores permite distribuir tareas y componentes dentro del sistema con las características de replicación, migración y monitoreo de forma aislada pero orquestada entre los diversos artefactos computacionales que lo componen. Este diseño es uno de los aportes principales de este trabajo.

De manera transversal existen dos elementos claves la seguridad basada en el modelo CIA (Consistency , Integrity , Aviability) y el proceso de gestión de nombres dentro del sistema, para la gestión de tareas, logs y manejo de información inter e intra capas. Finalmente el componente de resolución de nombres es transversal y constituye la entrada del Lenguaje de Programación TLÖN el cual manipula estas capas, y permite al usuario generar scripts para la ejecución de tareas sobre el sistema.

7.4. Esquema de la Virtualización

Para la primer capa de virtualización se tienen tres elementos principales para poder desplegar los servicios y realizar el mapeo de recursos sobre una red inalámbrica, estos artefactos computacionales son el Agente Local, El Orquestador y la abstracción Lógica del sistema Operativo, la cual es S.O.V.O.R.A. como se ve en 7-3. Estos artefactos computacionales comparten información de los estados loca-

les y globales del sistema, entrando en sincronía con los elementos descritos por el teorema CAP, el consenso y la necesidad de tener bases de datos distribuidas.

Cada uno de los agentes locales manipula los contenedores que son otro artefacto computacional clave en la virtualización para aislar aplicaciones, administrar recursos de acuerdo a las políticas indicadas por el Orquestador y operar micro servicios. En cada uno de los nodos existe uno o más agentes, del mismo modo el orquestador dependiendo las necesidades de aplicaciones y servicios solicitados por los usuarios, el orquestador toma decisiones con base en las políticas pseudo-sociales programadas en él o creadas por el usuario y opera los recursos para generar la siguiente abstracción de la capa de virtualización.

El modelo de aplicaciones esta desarrollado bajo la arquitectura de microservicios y es gestionada sobre contenedores para permitir su administración desde un host hasta la nube.

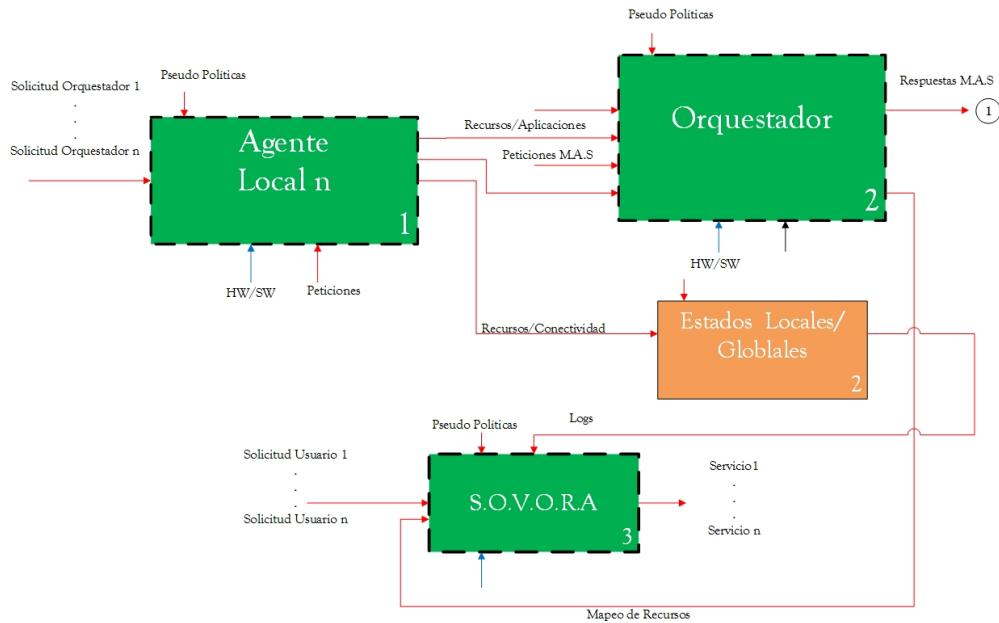


Figura 7-3: Modelo Virtualización

Este módulo contiene la información de los estados globales y locales para generar la abstracción lógica de los recursos mapeados y de esta manera distribuir las aplicaciones sobre el sistema.

7.5. Modelo del Sistema Operativo

Una vez desplegados los recursos y la información de los contenedores, y con la ejecución de las pseudopolíticas del orquestador, se construye el Sistema Operativo Virtualizado Orientado a Redes

Ad hoc - S.O.V.O.R.A, este sistema es la abstracción donde los diferentes módulos son desplegados y los microservicios son soportados, como se ve en 7-4 donde existen agentes en los diferentes niveles para realizar las labores de monitoreo en cada uno de los niveles de despliegue de del sistema, infraestructura, contenedores y microservicios, desplegados para conocer el estado actual y alimentar el estado global historico del sistema, estas tareas son enlazadas por la entidad que haga las veces de orquestador.

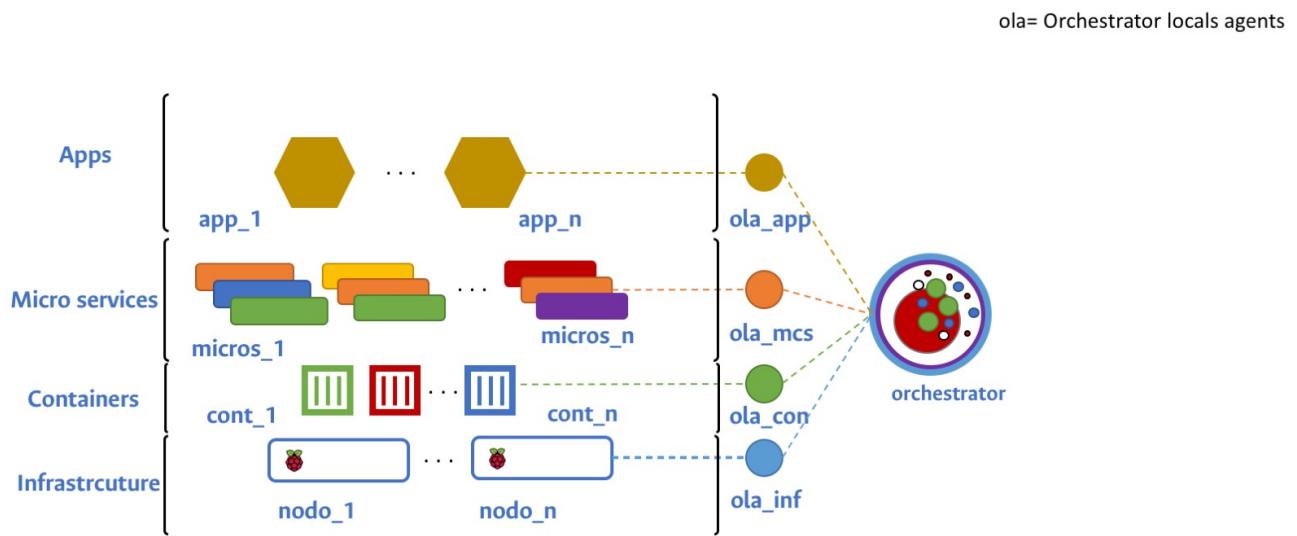


Figura 7-4: Niveles de S.O.V.O.R.A.

Finalmente se sigue un esquema de programación distribuida con unidades referenciadas a procesos, **Entidades de Estado Completo:** (Hilos, celdas, puertos, objetos), **Entidades de asignación simple:** (Variables de flujos de datos y streams) y **Entidades sin Estado:** (procedimientos, funciones, registros, clases), descritas en los siguientes diagramas.

7.5.1. Red Ad hoc -Pseudo Territorio

El clúster como unidad básica del Sistema Tlön, contiene los recursos necesarios para su despliegue. En este trabajo un clúster es una red ad hoc con un mismo SSID, como tal la red ad hoc es un sistema autoconfigurable , dinámico y estocástico, móvil e inalámbrico, esta red puede tener los roles

de cliente y/o enrutador en diferentes instantes de tiempo, los protocolos de enrutamiento permiten la auto gestión en la capa física de la misma, este comportamiento contribuye a conformar sistemas más dinámicos y escalables, como lo son las ciudades inteligentes y el Internet de las Cosas.

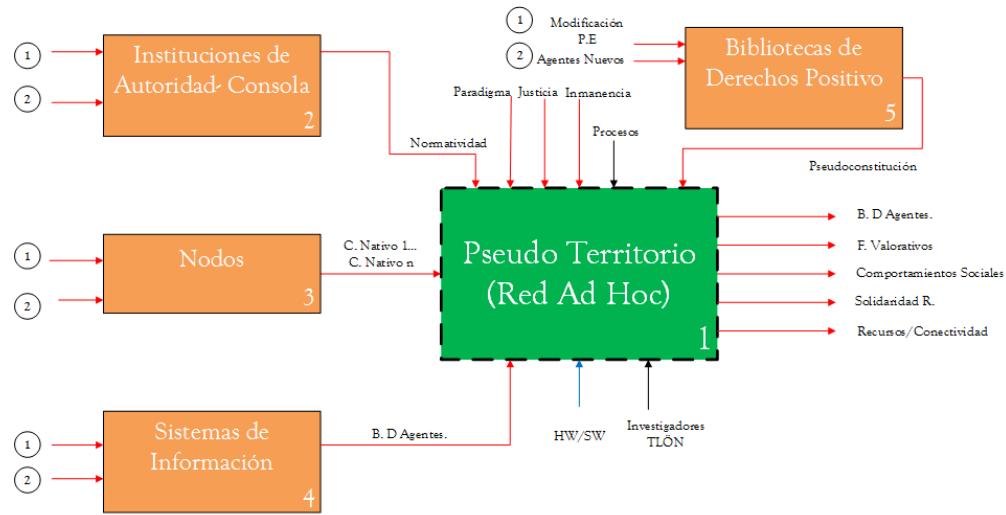


Figura 7-5: Red Ad hoc Pseudo Territorio

Bajo este modelo la red ad hoc es el pseudo territorio de l sistema, esta bajo el administración de las instituciones en este caso el orquestador y posiblemente el agente local si es dotado con los privilegios para gestionar la red, contiene el conjunto de políticas y es el medio de transmisión de la información de la base de datos distribuida, esta red opera en el modo MESH del estándar IEEE 802.11Committee y cols. (2012), mediante el protocolo de enrutamiento B.A.T.M.A.N. para la autoconfiguración de la red inalámbrica.

7.5.2. Diagrama Capa de Virtualización

Este diagrama esta compuesto por los tres elementos principales, el agente local, el Orquestador y la abstracción lógica del Sistema Operativo SOVORA. Donde se definen las entradas del sistema, un papel importante lo tiene la información consolidada por el orquestador actualizados los estados individuales de los elementos del sistema, para generar los estados globales administrados por el Orquestador, este diagrama se ve en la figura 7-6

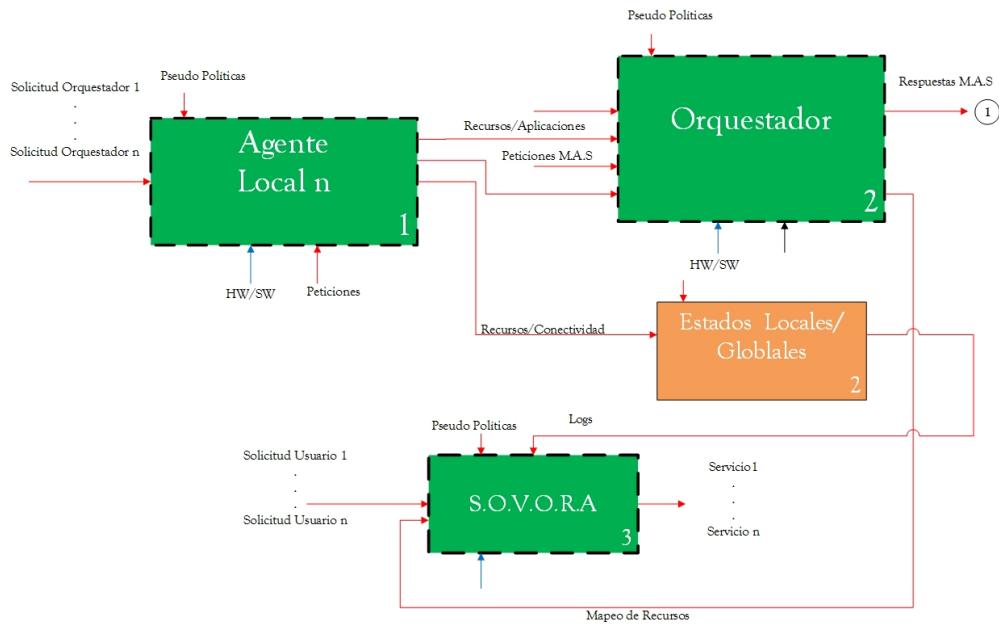


Figura 7-6: Modelo Virtualización Inalámbrica

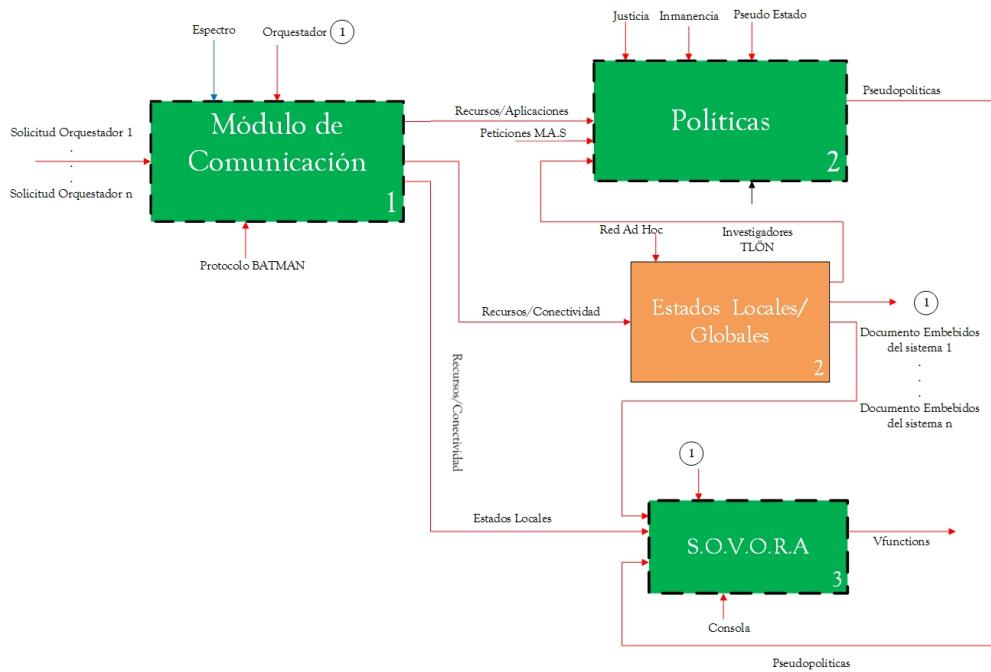
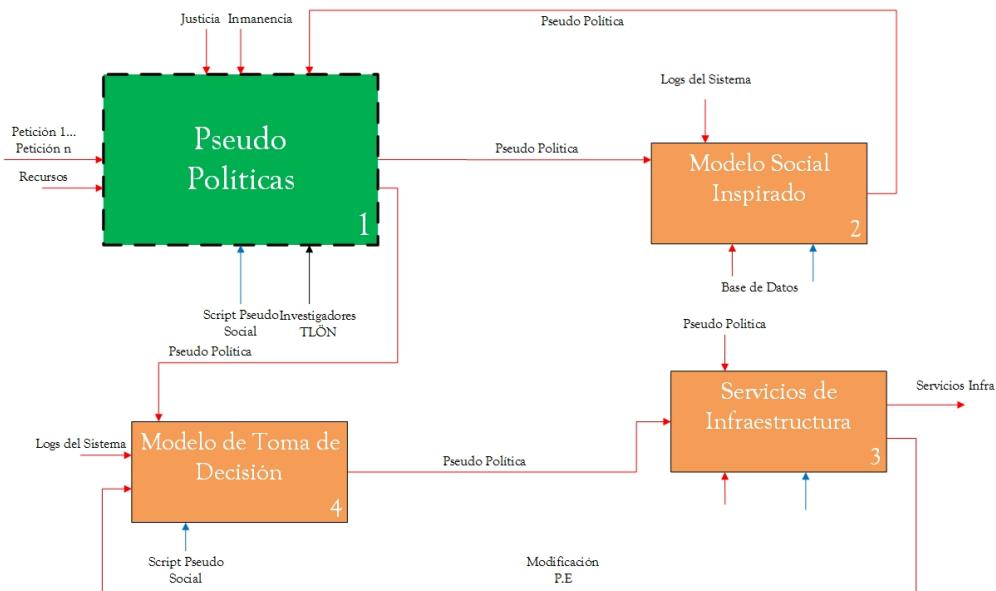
7.5.3. Diagrama Orquestador

El Orquestador se convierte en un elemento esencial dentro del sistema, como se evidencia en la figura 7-7, tiene la composición de un agente, este a su vez contiene la esencia del modelo pseudo social, con las pseudo políticas programadas en él, las cuales son activadas de acuerdo al modelo de toma de decisiones incluido en su estructura, tomando como entrada los logs del sistema proporcionado por la base de datos distribuida en todos los contenedores y microservicios asociados, las interacciones de los elementos del sistema son manejadas y controladas en esta instancia.

El elemento esencial dentro del orquestador e incluso dentro de un agente local dotado con mayor autonomía son las políticas las cuales permiten al orquestador ser dotado de la capacidad de decisión con base en la política seleccionada para resolver el problema seleccionado, ejecutar una aplicación o indicar los recursos presentados al usuario o reservados para la ejecución de una aplicación dentro del sistema.

7.5.4. Diagrama Agente Local

El agente local es uno de los componentes fundamentales de este proyecto, administra, monitorea y gestiona los recursos y aplicaciones usadas dentro del sistema, este elemento gestiona las comunicaciones del nodo con el orquestador, informa de su estado y mantiene un log completo de las

**Figura 7-7: Diagrama Orquestador****Figura 7-8: Políticas Orquestador**

interacciones con el ambiente base o en nuestro caso la red Ad hoc, el agente local es quien nutre la base de información del orquestador y reserva recursos para la ejecución de aplicaciones distribuidas en la red, del mismo modo entrega los resultados de la operación al o a los nodos que requieren la información, este artefacto computacional es base del sistema de detección de fallas.

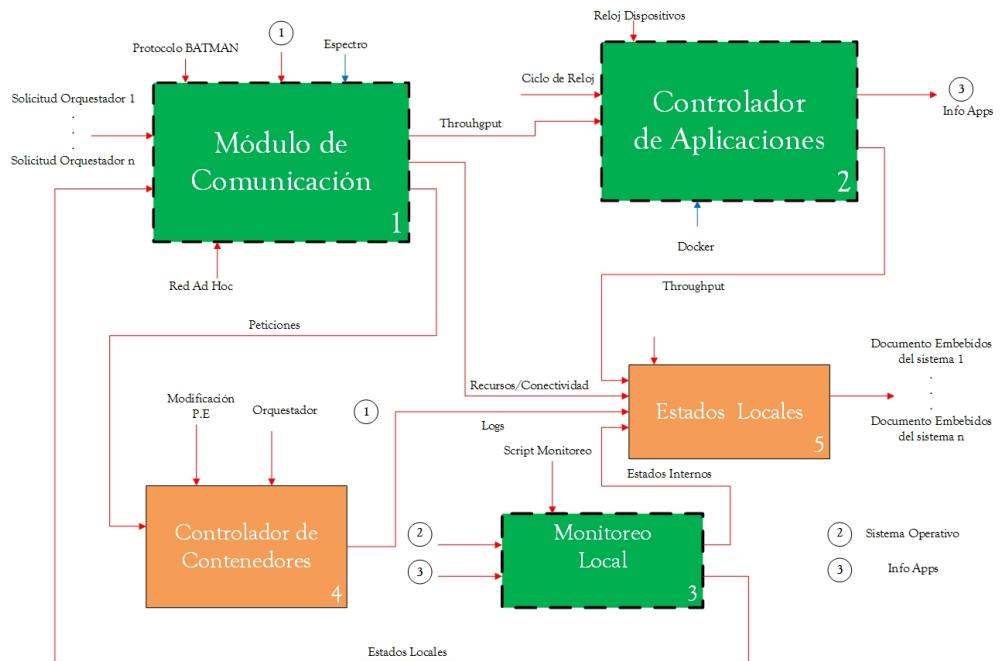


Figura 7-9: Agente Local

7.5.5. Diagrama SOVORA

Este diagrama constituye la más alta abstracción de la capa de virtualización donde se tienen los recursos y las funciones básicas, para la operación de las capas que se desplegaran sobre la virtualización, algunas de estas funciones virtuales son:

- Shell de TLÖN.
- Comandos para la distribución de hilos, memoria, disco duro y comunicaciones.
- Comando Información general del sistema.
- Editor de Texto del Sistema.

- Comandos para la configuración del Orquestador.
- Consola de Monitoreo.
- Primeras Instituciones de TLÖN- Control de Recursos

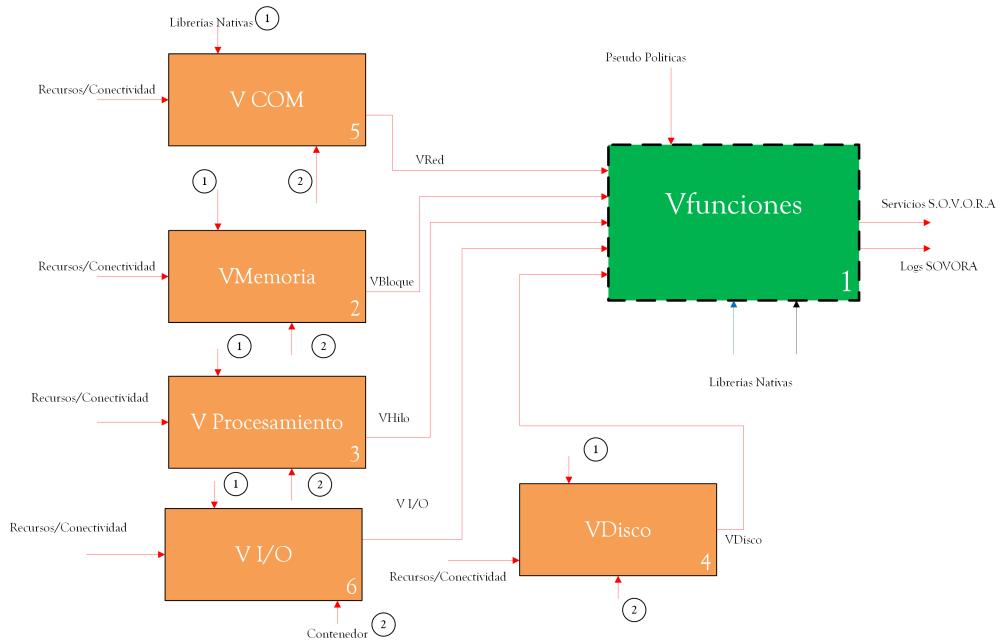


Figura 7-10: Diagrama S.O.V.O.R.A.

El control principal también viene dado por el Orquestador el cual maneja varios dominios para operar el sistema y es clave en la administración del sistema, microservicios y las aplicaciones, en esta abstracción se toman los estados globales, para gestionar el sistema y, mantener estables los servicios y los pedidos de las capas superiores.

7.5.6. Primitivas del Sistema Virtualizado

Las primitivas de este sistema están orientadas a sistemas distribuidos y elementos de la virtualización, teniendo en cuenta la distribución de recursos, los esquemas de comunicación necesarios y la distribución de tareas en el sistema hacen necesarios la identificación de elementos principales que denominaremos objetos.

Primitivas de S.O.V.O.R.A.

- **Nodo (Node):** Este objeto representa un dispositivo particular
- **GGlobal(Ggroup):** Este objeto representa un conjunto de nodos, un grupo Global
- **Grupo(Group):** Este objeto representa un conjunto de Agentes Locales
- **Agente Local(Agl):** Este objeto representa un agente con capacidades de comunicación, gestión y monitoreo dentro del nodo. En un nodo pueden existir uno o más agentes locales
- **Orquestador(Orch):** Este objeto permite acceder a los estados globales del sistema, desplegar políticas y distribuir tareas sobre el sistema
- **Institución(Inst):** Este objeto permite crear módulo de control dentro del sistema y adoptar políticas dentro del sistema.

Para el manejo de ese sistema las primitivas propuestas son las siguientes:

- a) **A nivel de Nodo:** Se generan las siguientes instrucciones para conocer los estados del nodo y los elementos necesarios para :

`is_alive() [opciones - node, ip]`

Permite saber si un nodo esta en línea y conectado en el sistema

`node() [opciones - node, ip]`

Da información actual del nodo, información de recursos, ip y UUID

`process() [opciones - node, ip]`

Indica los procesos del sistema corriendo en el nodo

- b) **A nivel de Orquestador:** En este nivel se crean comandos propios del Orquestador para trabajar en los dos niveles que le corresponden:

`sstatus [opciones] [ip, port]`

Indica el estado actual del sistema

nstatus [opciones] [ip, port]

Indica el estado de la red del sistema

- c) **A nivel de SOVORA:** En este nivel los comandos propuestos para las funciones básicas son los siguientes:

server [opciones] [ip, port]

Crea un servidor multihilo en un nodo particular.

global() [opciones - ssid]

Indica los miembros de un clúster físico.

org() [opciones - ssid]

Indica la organización de los nodos en la red.

gglobal() [opciones - nodes]

Crea un clúster lógico, un grupo de agentes locales.

disk() [opciones - app]

Reserva un espacio de disco distribuido.

cpu() [opciones - app]

Reserva espacio de ejecución (Hilos).

mem() [opciones - app]

Reserva memoria distribuida para aplicaciones.

io() [opciones - app]

Reserva dispositivos de entrada y salida para aplicaciones.

pidg() [opciones - nodes]

Muestra los procesos actuales en el sistema.

dprocess() [opciones -node,ip]

Crea un procesos en un nodos específico.

mess() [opciones]

Crea un un mensaje entre nodos del sistema.

clock() [opciones]

Establece el reloj del sistema.

7.5.7. Instituciones del Sistema

Para implementar el sistema social inspirado se propone el diseño de la institución como artefacto computacional, el cual puede ser operado por un agente, este modelo permite la creación de módulos para el control del sistema o crear funciones particulares a partir de las políticas y la validación de los estados globales y locales, tanto de los nodos como de los agentes locales.

El despliegue de las políticas y las funciones, están enlazadas con las del orquestador, quien provee las políticas, dentro de las instituciones están los controles de datos por las leyes naturales y positivas. La función de las instituciones esta dada por el módulo de operación que indica que elemento se va a monitorear, de esta manera se reciben las solicitudes del usuario, aplicaciones y de otros módulos del sistema.

Estas instituciones son apoyo al orquestador que toma información de estas instituciones para el despliegue de políticas, algunas de estas instituciones pueden por ejemplo denegar servicios a un nodo, un agente o una comunidad de agentes. El modelo propuesto se ve en la figura 7-11

7.5.7.1. Generador de Instituciones

Uno de los elementos claves en este diseño es la inclusión de institucionalidad en las interacciones entre los servicios, aplicaciones y el sistema operativo. El desarrollo de este trabajo esta orientado a la creación de agentes artificiales, son ellos quienes proveen servicios y operan las instituciones, pero en si una institución es un conjunto de recursos provisionados para una función específica a nivel del sistema multi agente. Este modelo es uno de los aportes de este trabajo.

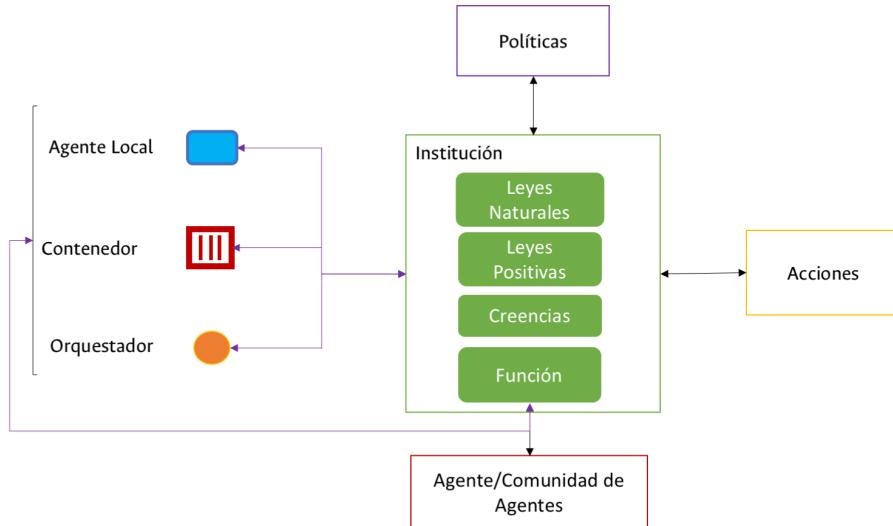


Figura 7-11: Modelo Instituciones

Una institución puede ser definida así:

$$I = \langle \mathcal{R}, \mathcal{N}, \mathcal{A}_r, \mathcal{A}_c, \mathcal{H}, \mathcal{G}, \mathcal{M} \rangle \quad (7-1)$$

Donde \mathcal{R} es el conjunto de recursos necesarios para su operación, \mathcal{N} esta compuesto por el conjunto de \mathcal{L}_N que es el conjunto de leyes naturales dadas por el Estado y \mathcal{L}_P las leyes positivas propuestas por el sistema multi agente, \mathcal{A}_r representa el artefacto que posee la función de dicha institución y \mathcal{A}_c las acciones llevadas a cabo por la institución, \mathcal{H} es la historia, \mathcal{G} representa el conjunto de Agentes que operan la institución, finalmente \mathcal{M} representa el conjunto de mensajes o esquema de comunicación usado por la institución para realizar las funciones y comunicarse con los agentes.

La estrategia definida para el generador de instituciones es recibir la solicitud del Sistema Multi agente, contenida en la ecuación 7-1, estos parámetros son usados para reservar la cantidad de recursos, la ubicación de la institución y las necesidades de comunicación del módulo y/o microservicios desplegado.

Un modelo similar puede verse en Tomic y cols. (2018), donde se presenta una formalización del modelo en tres niveles, esta propuesta al igual que el modelo del sistema Tlon esta orientada a MAS, de esta manera la contribución del autor mencionado esta basada en la analogía de un juego de fútbol donde incluyen jugadores, arquero, un referí y una audiencia (roles) un campo (artefacto) , todos

actúan bajo un conjunto de reglas (actos), los tres elementos propuestos en el modelo mencionado son:

$$Artefacto = \{art_1, art_2, \dots, art_n\} \quad (7-2)$$

$$roles = \{rol_1, rol_2, \dots, art_n\} \quad (7-3)$$

$$Actos = \{act_1, act_2, \dots, act_n\} \quad (7-4)$$

Teniendo en cuenta esta premisa y el modelo propuesto de instituciones en este trabajo esta orientado al desarrollo de los Artefactos \mathcal{A}_r , como una unidad básica de servicio dentro del sistema, las leyes indicadas se encuentran tanto en las políticas como en el conjunto de comportamientos establecidos a un agente dependiendo su rol en el sistema, en este caso el agente local y su relación con la entidad orquestador, este modelo base de este desarrollo contiene una serie de relaciones descritas en la sección de formalización (Bigrafos), pero adicionalmente como base del sistema multiagente. Es necesario entonces indicar que es un norma en el sistema propuesto.

Normas en S.O.V.O.R.A.

Una norma esta descrita como una relación entre dos o más estamentos de una institución que indica que acciones podría o no podría hacer un miembro sobre el sistema, esta definida por un cualificador, en el contexto del sistema TLÖN es el conjunto de verbos instanciados en los diferentes tiempos verbales presente, pretérito imperfecto, pretérito perfecto simple, futuro, este calificador denominado como t , esta acompañado de los artefactos , roles y acciones , definidos como ins :

$$inst : artefacto \times Acciones \times (artefacto \times roles) \quad (7-5)$$

mientras que las normas están descritas como :

$$Normas = \mathcal{L}_{\mathcal{N}} \cup \mathcal{L}_{\mathcal{P}} \quad (7-6)$$

El sistema TLÖN como se ha visto esta diseñado y desarrollado bajo un modelo de capas y elementos transversales, estos elementos son dominios que están inter relacionados y forman interacciones entre las instituciones definidas como:

$$\mathcal{D} = \langle \mathcal{G}, \mathcal{I}, \mathcal{N}, \mathcal{B}, \mathcal{R} \rangle \quad (7-7)$$

\mathcal{G} es un conjunto de agentes

\mathcal{I} es un conjunto de instituciones

\mathcal{N} es un conjunto de normas

\mathcal{B} es un conjunto de comportamientos

\mathcal{R} es un conjunto de recursos

Sin embargo el alcance de este trabajo esta orientado a el manejo y asignación de recursos dentro del sistema para las aplicaciones distribuidas desplegadas, esto traducido en instituciones y recursos virtuales reservados para ellas.

7.5.7.2. Dispersión y Recopilación

El Modelo de dispersión y recopilación de contenidos, aplicaciones o agentes, esta relacionado con el concepto de inmanencia visto en la sección 5.1.2, asociado a la dispersión y recopilación de contenidos se tiene el tamaño de la porción o pieza a distribuir, sobre una ambiente distribuido como la red ad hoc es posible construir aplicaciones particionadas o distribuidas como contendores, o incluso realizar la partición de una imagen antes de ser ejecutada, en cuanto a los agentes el modelo aplica del mismo modo si se define un agente como una máquina de estados finitos $\mathcal{M} (\Sigma, \Gamma, S, s_0, \delta, \omega)$, donde: Σ es el alfabeto de entrada, Γ es el alfabeto de salida, S es u conjunto finito de estados posibles. s_0 es el estado inicial del agente. δ es la función de transición $\delta : S \times \Sigma \rightarrow S$ $\delta : S \times \Sigma \rightarrow S$. Por us parte ω es la función de salida.

Es necesario dentro de este modelo y dependiendo de la cantidad de nodos del sistema hacer copias de la información para agregar redundancia y alimentar el sistema de detección de fallos de la sección 7.5.7.7, este módulo en particular puede ser representado por la función Binomial negativa o de Pascal Mood (1950).

Definición 7.5.1 (Distribución Binomial Negativa) Una variable aleatoria X con densidad:

$$f_X(x) = f_X(x; r, p)$$

$$\begin{cases} \binom{r+x-1}{x} p^r q^x = \binom{-r}{x} p^r (-q)^x & \text{para } k = 0, 1, 2, \dots \\ 0 & \text{en otro caso} \end{cases} \quad (7-8)$$

$$\binom{r+x-1}{x} p^r q^x I_{0,1,\dots}(x)$$

donde los parámetros r y p satisfacen $r = 1, 2, 3, \dots$ y $0 < p \leq 1$ ($q = 1 - p$), es definida como distribución binomial negativa. Si en la distribución binomial negativa $r = 1$, entonces la densidad binomial negativa

se especializa en la densidad geométrica

Para utilizar este modelo es necesario usar el proceso de truncamiento para indicar el número finito ya sea de nodos o de recursos disponibles para dispersar y recopilar los archivos, módulos, imágenes y demás artefactos de cómputo distribuidos en la red.

La distribución binomial negativa, como la Poisson tienen los enteros n negativos para sus puntos de masa, así la distribución binomial negativa es potencialmente un modelo para experimentos aleatorios donde se requiere hacer un conteo en algún orden, en nuestro caso los nodos y los recursos disponibles. Esta familia parámetrica de función de densidad discreta requiere de un proceso de truncamiento para poder expresar las limitaciones de recursos y de nodos en el sistema.

7.5.7.3. Agente Móvil

Los Agentes Móviles (MA), son un paradigma para la computación distribuida que permite generar un gran número de aplicaciones potenciales, se puede decir entonces que un Agente Móvil es una pieza de código independiente de la plataforma y del sistema operativo Cao y Das (2012) con características específicas que le permiten moverse, actuar y ejecutarse en nodos en los que este no ha sido creado. Se diferencia de otro tipo de agentes que no tienen movilidad ya que no solo la información, sino el mismo código fuente se transporta de un nodo a otro proporcionando flexibilidad y escalabilidad a la aplicación desarrollada.

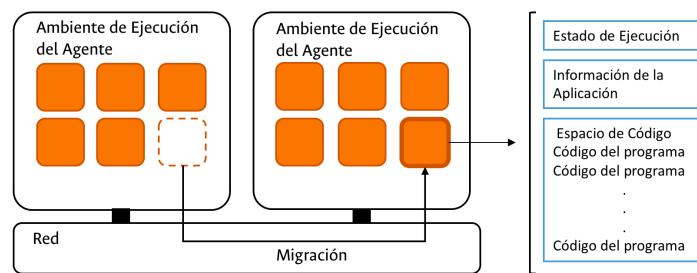


Figura 7-12: Modelo Agente Móvil

Al igual que un agente tradicional requiere de un ambiente de operación que interactúe con él y con el sistema en general, en la figura 7-12 se muestra un modelo base de operación de un Agentes Móviles, la primitiva básica es la migración que le permite moverse a través de la red conservando tres elementos, el estado de la aplicación, la información de la aplicación y el código de ejecución, esto con el

objetivo de mantener las tareas en ejecución sin interrupción y disminuir la carga de las transmisiones, como las generadas por el esquema cliente-servidor.

Tres acciones particulares son propuestas bajo este modelo, la clonación del agente, la migración y la destrucción. Estos estados en conjunto hacen posible la movilidad del agente siempre y cuando exista el ambiente base para su despliegue, estas actividades son desarrolladas tanto en ubicaciones físicas como lógicas, lo cual permite una flexibilidad a lo largo del sistema si hay una redistribución de recursos, o si algún nodo desaparece del sistema.

7.5.7.4. Balanceador de Carga

Para realizar una distribución de servicios, ser capaces de realizar las tareas de dispersión y recopilación, operar el agente móvil y distribuir tareas como la detección de fallas o generar rutinas de seguridad sobre el sistema, es necesario diferenciar el tráfico y las tareas asignadas, por ello dentro de este modelo es necesario realizar un modelo de balanceo de carga, orientado a la distribución de flujos, bajo esta estrategia es posible sobre redes ad hoc generar un plano de control y uno de datos para gestionar las tareas de control, señalización y monitoreo dentro del sistema, diferenciándolos de los flujos de aplicaciones y servicios prestados al usuario.

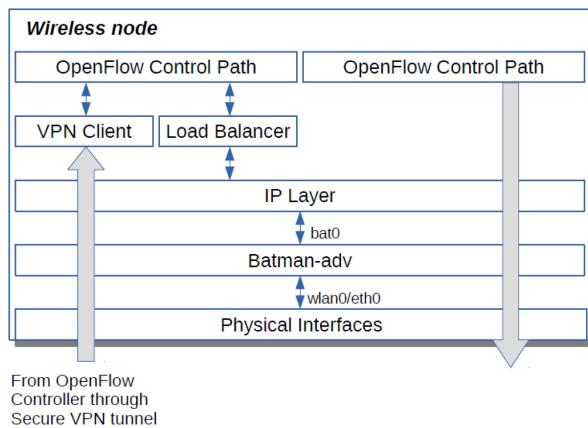


Figura 7-13: Modelo Balanceador

El modelo propuesto se puede ver en la figura 7-13, este modelo cuenta con el motor de OpenFlow McKeown y cols. (2008) para diferenciar los flujos de tráfico y operar sobre el protocolo BATMAN en nuestro caso, este modelo incluye el balanceo de carga sobre los flujos y la diferenciación de tráfico de control y datos.

7.5.7.5. Seguridad Virtualización

La idealización de un modelo distribuido con características de movilidad y auto organización generan una superficie de ataque mayor, vectores de ataques adicionales y un conjunto de amenazas y vulnerabilidades superiores respecto a los sistemas tradicionales. Es por ello que un sub sistema de seguridad es necesarios sobre este sistema distribuido propuesto y más aún en nuestra capa de abstracción de sistema operativo, es por ello que se propone el uso del modelo CIA (confidencialidad, integridad y disponibilidad) Perrin (2008) el cual enmarca los requerimientos de seguridad en tres características fundamentales, las cuales deben ser cubiertas en toda la superficie de ataque del sistema y desarrolladas en diversos módulos. Para el modelo propuesto se distribuye el sistema como se ve en la figura 7-14

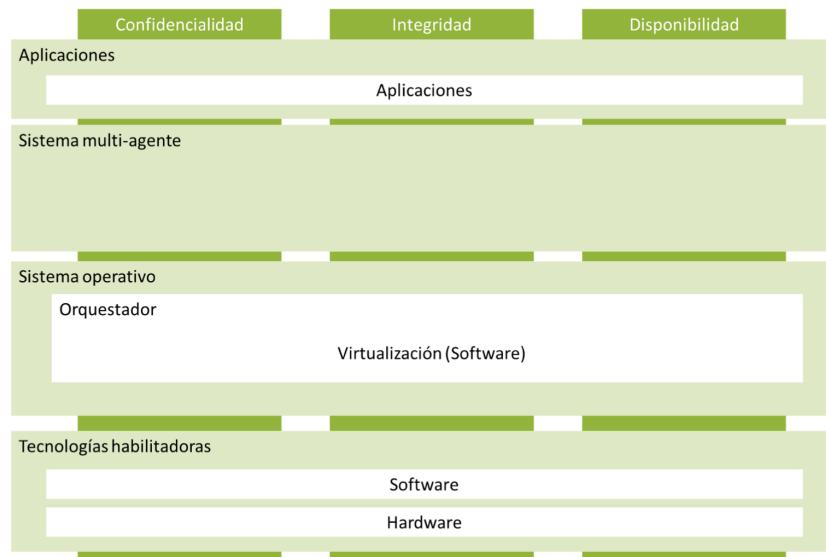


Figura 7-14: Modelo Seguridad Propuesto (CIA)

Las operaciones propuestas son en dos niveles el primero gestionado por el orquestador quien tiene las políticas de seguridad del sistema e indica las acciones a los agentes locales, el siguiente nivel es la operación autónoma del agente local para la ejecución de las políticas de seguridad sobre cada nodo, con información hacia el orquestador reduciendo la cantidad de mensajes y trazas de control sobre el sistema, esto depende de la cantidad de nodos sobre el sistema y servicios.

7.5.7.6. Sistemas de Archivos

Existen diversos modelos de almacenamiento en red, pero los métodos clásicos son sin duda alguna dos, NAS y SAN Jepsen (2003), NAS (Network Attached Storage) consiste en un gestor de archivos agregado a la red el cual administra los accesos a la información almacenada, bajo este esquema un solo nodo puede ser el servidor que contiene los archivos un modelo más centralizado, otro modelo es el de SAN(Storage Area Network) que utiliza accesos compartidos de equipos y usuarios mediante el protocolo Fibre Channel (FCP) en una arquitectura basada en switches o arreglos de discos para el almacenamiento, creando una red alterna de gestión para el almacenamiento de datos en el sistema.

En nuestro esquema es importante indicar que es la virtualización de almacenamiento, básicamente es el acceso a volúmenes virtuales, los cuales a su vez son un conjunto de unidades de discos o piezas de discos físicos distribuidas las cuales son gestionadas como una simple entidad, como se ha descrito en el capítulo 3, este volumen lógico puede cambiar de tamaño y ser gestionado por un gestor de volúmenes como pieza de software o por un componente de Hardware como un controlador RAID (Redundant array of independent disks), el cual requiere más de un disco para hacer las operaciones de control de datos y de discos virtuales.

Dentro del esquema es necesario tener en cuenta otros dos procesos independientes del proceso de Almacenamiento, el primero el respaldo o *backup* y el segundo derivado del primero la restauración. Estos procesos propuestos están relacionados ampliamente con el proceso de dispersión y recopilación enunciado en la sección 7.5.7.2, es decir la gestión de copias o en nuestro caso la migración. La distribución sobre los volúmenes de disco mapeados están basados en el proceso de Dispersión y Recopilación de los contenidos, es importante recordar que este modelo tiene la funcionalidad de hacer copias de la información, al igual que este modelo está orientado a sistemas no persistentes de almacenamiento de datos, sin embargo se puede extender a volúmenes.

El modelo propuesto es mediante el uso de un gestor de volúmenes virtuales el cual es operado por el orquestador quien recibe el espacio reservado por el Agente local para ser usado y mapeado dentro del Sistema, este proceso se realiza de manera análoga como se hace con los otros recursos. Este modelo se realiza con el protocolo NFS (Network File System) y SFTP (Secure File Transfer Protocol) para la transmisión de archivos y reducir la carga sobre la red, este módulo es ampliamente relacionado con el módulo de balanceo de carga para evitar comportamientos no deseados en la red y reducir los índices de Disponibilidad de la información y por supuesto de operación del sistema. De este modo se utiliza el esquema FUSE (FileSystem in User space - Sistema de archivos en el espacio de usuario), durante mucho tiempo, la implementación de un sistema de archivos de espacio de usuario se consideró imposible, el desarrollo hecho con FUSE como un módulo kernel que admite la interacción entre VFS del kernel y aplicaciones de usuario no privilegiadas con un API a la cual se accede desde el espacio de usuario. Al usar esta API, se puede escribir cualquier tipo de sistema de archivos usando casi cualquier lenguaje gracias a la interoperabilidad de este sistema.

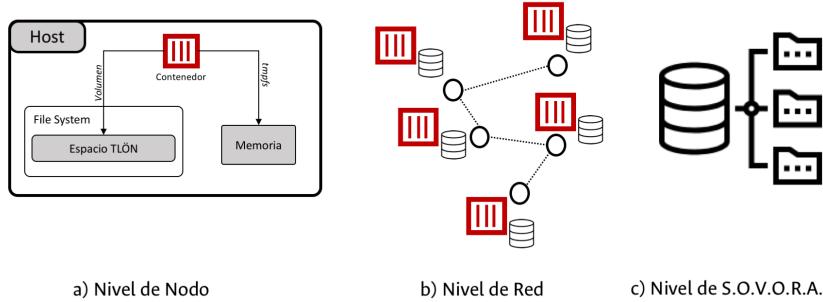


Figura 7-15: Modelo de Disco Distribuido

El modelo propuesto para la gestión de almacenamiento es crear volúmenes virtuales asociados a los contenedores, es decir la información sera almacenada en un espacio específico del *file system* del dispositivo, cada volumen es mapeado por el agente local y disponible en el sistema para información no persistente continuara en el espacio de directorios temporales *tmpfs* almacenados en la memoria del dispositivo, el uso de volúmenes de docker permite al sistema:

- Los volúmenes son más fáciles de respaldar o migrar que las unidades montadas.
- Los volúmenes funcionan tanto en contenedores de Linux como de Windows.
- Los volúmenes se pueden compartir de forma más segura entre varios contenedores.
- Los controladores de volumen le permiten almacenar volúmenes en hosts remotos o proveedores de la nube, cifrar el contenido de los volúmenes o agregar otras funciones.

A este tipo de almacenamiento asociado a contenedores se les asigna la cuota, directorio raíz o punto de montaje, el driver seleccionado para el montaje es *nfs*, lo cual nos permite asociar la *IP* del dispositivo para el acceso a este recursos, permitiendo crear un sistema distribuido de disco con este estándar abierto como se puede ver en el siguiente comando de docker.

```
$ docker volume create --driver local \
```

```
--opt type=nfs \
--opt o=addr=169.254.1.1,rw \
--opt device=/home/pi/tlon \
foo
```

Esta información de volúmenes *nfs* es compartida entre los agentes locales y el orquestador realizando el mapeo de los volúmenes disponibles en el sistema, creando un sistema distribuido de almacenamiento sobre la red ad hoc, como se ve en la figura 7-15, cada uno de los volúmenes se encuentra en dentro de un espacio reservado en el *file system*, la información recurso mapeado por el agente local es enviada al orquestador y de esta manera se tiene un consolidado de la cantidad de recursos disponibles en el sistema.

La creación de este volumen virtual esta a cargo del agente local para ellos se usa el servicio de gluster FS, esta herramienta dota al agente local de la capacidad de crear un File system distribuido bajo el concepto de "bloque", la gestión de bloques de disco en los diferentes nodos permite conformar un volumen virtual, y crear un File system dentro del sistema virtualizado.

El volumen es la colección de cada bloque reservado en cada nodo la gestión de estos espacios de disco están a cargo del gestor de gluster pero pueden ser operados por el agente local. Un volumen distribuido de Gluster-fs permite crear volúmenes de escalamiento de almacenamiento, mejora de rendimiento o mixtos cada bloque esta distribuido dentro de los diferentes nodos como bricks o bloques que conforman el *filesystem* inicialmente en este implementación no existe redundancia dentro del sistema pero usando el servicio de dispersión y recopilación propuesto es posible solventar la falta de redundancia. El manejo de los bloques es bajo el esquema de volúmenes *nfs* del mismo modo en los volúmenes de las aplicaciones dentro de los contenedores, por lo cual existen dos espacios de datos dentro del sistema los volúmenes de aplicaciones de docker y el volumen del filesystem del sistema expresados en bloques en cada nodo.

La comunicación entre el módulo de kernel de FUSE y la biblioteca de FUSE denominada libfuse se realiza a través de un descriptor de archivo especial que se obtiene al abrir el directorio /dev/fuse, un modelo que muestra este proceso de interacción entre los espacios de kernel y usuario se puede ver en la figura 7-16.

7.5.7.7. Detector de Fallas

En sistemas distribuidos uno de los objetivos fundamentales es conocer el estado de los miembros del sistema y de los servicios desplegados en el mismo, un sistema de detección de fallas es necesario para poder monitorear los componentes, detectar fallas, notificar y registrar los eventos ocurridos cada vez que ocurra una falla o se detecte una operación incorrecta de los mismos Hayashibara y cols.

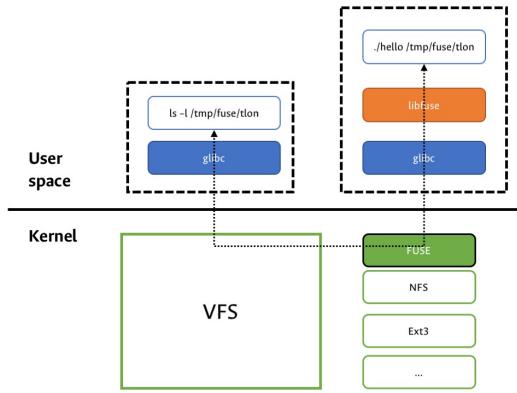


Figura 7-16: File System espacio de Usuario

(2002).

Las bases de los sistemas de detección de fallas fueron introducidas por Chandra y Toueg Chandra y Toueg (1996), hoy en día vigentes, presentan con claridad el problema del monitoreo, el consenso y la toma de decisión en las acciones así como las estrategias de monitoreo en un sistema distribuido.

Teniendo en cuenta la naturaleza pro activa tanto del protocolo BATMAN, como del agente local e incluso del orquestador el esquema propuesto es el de extracción donde el detector de fallas es activo y el componente pasivo, conocido como el *Pull Model*, en este modo de operación la carga en la red se reduce y depende de la cantidad de solicitudes realizadas a cada componente del sistema enviadas por el monitor, en este caso el monitor es el Orquestador, componente del sistema de monitoreo. El problema radica en que el orquestador solo puede sospechar de una falla después de la llegada de un mensaje de respuesta a una solicitud de estado sin embargo, el monitor no puede sospechar o detectar el falla de un componente hasta después de que le envíe una solicitud de liveness.

Este modelo de operación funciona bajo el concepto de *Agente Local → Orquestador* realizado en este trabajo y expuesto con más detalle en la sección 7.8, en resumen se pude observar el modelo desarrollado en la figura 7-17. Este es uno de los aportes de este trabajo.

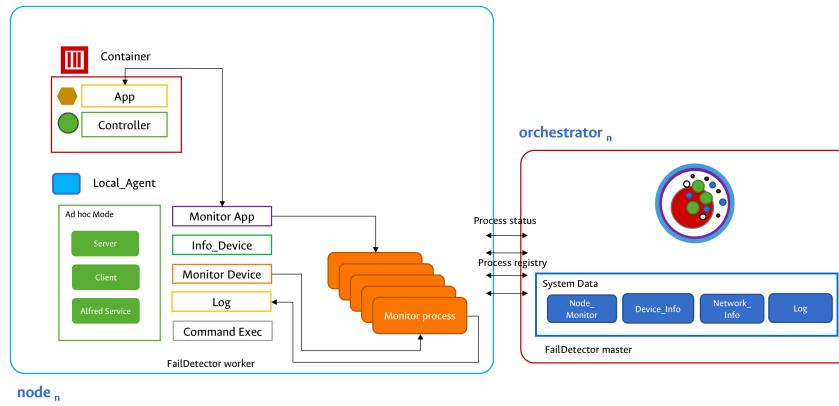


Figura 7-17: Detector de Fallas propuesto

7.5.7.8. Socket para Sistemas Distribuidos

Uno de los problemas típicos en la capa de transporte son las restricciones de estructuras básicas como el socket que requiere únicamente la dirección de destino y el protocolo para definir un servicio, variables como el ancho de banda y las necesidades de la aplicación en conectividad y el uso de diversos protocolos, salen del dominio de un socket clásico. Por ello es necesario en sistemas distribuidos dinámicos una nueva versión de socket adaptable, un modelo base puede ser el Socket Intent Tiesel y cols. (2018), un tipo de socket adaptable a las aplicaciones, es decir tiene la capacidad de manejar el ancho de banda, la latencia y costo de transmisión. El hecho de ser adaptable a las tasa de bytes, y más importante aún la capacidad de conocer la disponibilidad de rutas y de paquetes en el segmento asignado, permite la adaptabilidad a diferentes protocolos y servicios, permitiendo al desarrollador explotar estas características en el sistema operativo y en las aplicaciones.

Estos elementos son categorizados como *preferencias, expectativas, características y resiliencia*, las cuales son explotadas de formas novedosas para definir las necesidades de transporte en el despliegue de la aplicación y por ende dar un grado más de libertad para el desarrollo de aplicaciones, del mismo modo permite la conectividad con varias redes, sin embargo deben agregarse los elementos asociados al modelo de socket seguro, para de esta manera brindar una abstracción de comunicación robusta, confiable y segura para el despliegue de aplicaciones en sistemas distribuidos.

7.5.8. Dispositivos de Entrada Salida

El manejo de dispositivos de entrada y salida siempre está sujeto al nivel de operación requerido por la aplicación, estos pueden estar en tres niveles, sobre el hardware, el controlador del dispositivo o una aplicación de software, esta distribución se puede ver en la figura 7-18.

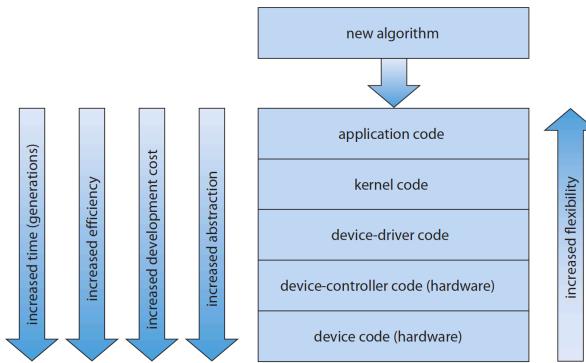


Figura 7-18: Modelo de Funcionalidad de dispositivos I/O Silberschatz y cols. (2014)

Una nivel de operación de dispositivos de I/O es justamente la implementación de algoritmos en el ámbito de aplicaciones donde se pueden identificar errores de operación y bloqueos del sistema, el desarrollo de estos algoritmos requieren reinicio siempre y cuando existan cambios en el código, este enfoque no es el más eficiente porque no explota las funcionalidades del kernel del sistema, sus estructuras de datos o su mismo funcionamiento. Por su parte el desarrollo de aplicaciones de medio nivel esta orientado al desarrollo de estructuras de datos y códigos directamente en el kernel del sistema, pero el desarrollo de estas aplicaciones requiere de un esfuerzo mayor y una depuración más rigurosa para evitar que el sistema completo colapse.

El alto rendimiento puede ser obtenido mediante una implementación especializada directamente en el Hardware, el dispositivo o incluso en el controlador, pero el tiempo de desarrollo, la identificación de errores es mucho más compleja que en los modelos anteriores, del mismo modo el despliegue de aplicaciones a este nivel es costoso y requiere de bastantes pruebas y dispositivos para realizar test del código desarrollado.

El manejo de dispositivos de entrada y salida se realizó en este trabajo al nivel de aplicación para la manipulación de periféricos y Hardware de propósito general como sensores, antenas, entre otros. Este proceso se realizó con la técnica de mapeo descrita el capítulo 3

Para el mapeo de estos dispositivos existen dos técnicas una es el mapeo de Puertos I/O usado en su mayoría por procesadores intel donde un dispositivo es mapeado en una posición de memoria con un espacio dedicado adicional denominado puerto que genera una dirección virtual, esta contiene un conjunto de instrucciones específicas del micro controlador, en este sentido existe un carga menor de decodificación de direcciones discretas a lo largo del mapeo de dispositivos, sin embargo requiere de más instrucciones para su ejecución. En contra parte el mapeo de Memoria I/O comparte la mismas direcciones de memoria del dispositivo lo cual permite usar las mismas instrucciones de memoria para acceder al dispositivo I/O.

El mapeo de dispositivos de memoria esta basado en la lectura de los espacios reservados de memoria en los dispositivos, para ello se hace referencia a los espacio reservados de memoria I/O, en este mapeo esta incluido en las acciones del agente local desarrollado, este agente reactivo posee políticas para el mapeo y recibe solicitudes por parte del orquestador, esta relación mediante las políticas indica el conjunto de normas y restricciones que tiene el agente en su operación.

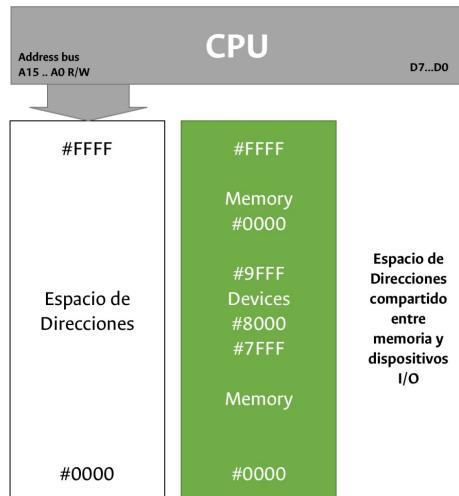


Figura 7-19: Modelo de Funcionalidad de dispositivos I/O Silberschatz y cols. (2014)

El mapeo de estas direcciones es realizado por el agente local y gestionada desde el orquestador o nodo con este rol, para la distribución de recursos y uso de los dispositivos de entrada y salida necesarios por el sistema.

7.5.9. Sistema Multi Agente

Dentro del modelo propuesto en el Sistema TLÖN es necesario integrar un modelo de inteligencia artificial como elemento base para desplegar aplicaciones, en este caso las aplicaciones están distribuidas sobre la abstracción de virtualización.

Para la realización de dichas aplicaciones se tienen en cuenta los conceptos de dispersión y recopilación para la manipulación de los agentes y la distribución de los mismos sobre la abstracción lógica. La métrica base de este modelo es la interacción al igual que en la representación con bigrafos, esta unidad nos permite definir con mayor precisión las necesidades de comunicación entre los agentes y las abstracciones lógicas del Sistema TLÖN.

Es necesario en este punto definir que es un sistema multi agente, Flasiński (2016) teniendo en cuenta

las siguientes características: *i*) un agente no puede resolver el problema en si mismo , *ii*) el sistema no es controlado de forma centralizada; *iii*) la data es distribuida en el sistema y *iv*) la comunicación en el sistema es realizada de forma asíncrona.

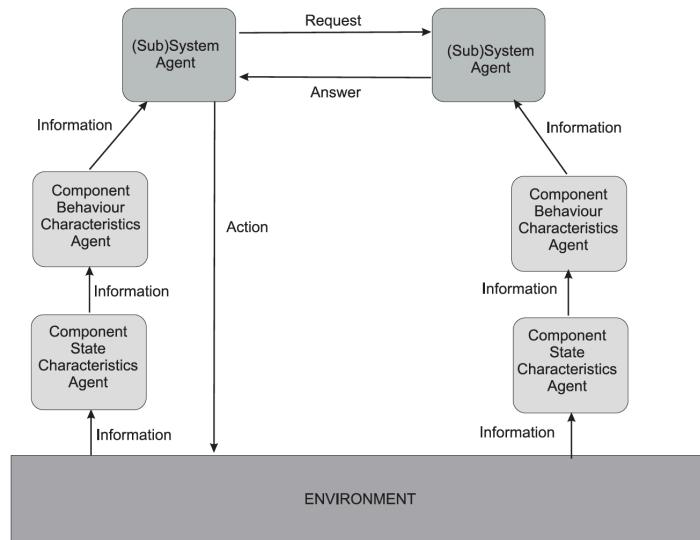


Figura 7-20: Modelo Sistema Multi Agente Flasiński (2016)

El problema clave radica en la comunicación entre los diferentes agentes diferenciando dos posibles estructuras una jerárquica y una realizada por medio de mensajes. El primer modelo es conocido como arquitectura de pizarra, donde existe un jerarquía para definir las interacciones de los agentes sobre el sistema, y contiene una clasificación de los agentes de menor a mayor jerarquía así: *agente reflejo*, *agente basado en modelos*, *agente basado en objetivos* y *el agente basado en utilidad*. Este último tiene una connotación sobresaliente donde la idea principal de un agente de este tipo esta basada en el concepto de *emociones*, al lograr la máxima satisfacción medida como una emoción positiva(como la indicada en la teoría de valoración de las emociones de Magda Arnold Arnold (1960)), al alcanzar este valor número o la representación asignada el grado de satisfacción es alcanzado, la mayor dificultad en el diseño y desarrollo de este tipo de agentes es atribuir valores numéricos a estos estados internos, que en ultimas representan las emociones.

En el segundo modelo basado en el intercambio de mensajes, el concepto base es la teoría del acto de habla inspirado por el concepto del *Sprachspielen* formulado por el filosofo Ludwig Wittgenstein Wittgenstein (2009), el cual presenta un modelo de interacción de agentes este se puede observar en la figura 7-20, este modelo se basa en las relaciones entre los agentes y los denominados *actos de comunicación*, por ejemplo una orden para hacer alguna acción, una promesa de hacer alguna acción o una solicitud de hacer algo, bajo estas interacciones deben existir unas reglas definidas las cuales garantizarían el desarrollo de las tareas enviadas mediante mensajes.

Es por ello que S.O.V.O.R.A. esta diseñado sobre el concepto de institucionalidad, siendo capaz de proveer las estructuras (instituciones) donde se desarrollen las interacciones y se definan las reglas de dichas interacciones entre los agentes que se comunican entre sí, en ultimas reglas sobre el uso de los recursos, el despliegue de las aplicaciones y el mantenimiento del sistema.

7.6. Descripción de las Pruebas

Para la validación de este modelo se realizó un banco de pruebas con seis dispositivos con recursos heterogéneos, Raspberry Pi 3 (4) (Raspberry Pi Zero w (1) y Odroid XU4 (1)), de esta manera se construyeron clúster lógicos y clúster físicos para el despliegue de SOVORA y las pruebas de operación del Sistema basado en los tres tipos de estado como se ve en la sección 8.3, finalmente se valida la operación del orquestador y la inclusión de políticas dentro del sistema para la gestión de los recursos.

El modelo de pruebas es la realización de la mayor parte de los modelos propuestos en las secciones anteriores, incluyen herramientas como Docker, B.A.T.M.A.N., Python, entre otras para el desarrollo del modelo, sin embargo como se ha visto en capítulos anteriores hay elementos restrictivos en estas herramientas para el desarrollo del sistema deseado.

La construcción del sistema inicialmente vista como un sueño, comienza a ser real a partir de la construcción de artefactos computacionales integrados en un sistema distribuido, sobre una red inalámbrica, la elección de las abstracciones correctas, la representación de las mismas y la ejecución en ambientes reales dan luces de la posible arquitectura del Sistema de Cómputo Social Inspirado TLÖN.

7.7. Componentes Conceptuales del modelo propuesto

En esta sección se reúnen todos los diseños realizados en el capítulo 7, se pasa de los modelos abstractos a la realización de los artefactos computacionales, dos elementos fundamentales en este desarrollo son el *agente local* y el *orquestador*, en conjunto estos dos artefactos generan el mapeo mencionado en el capítulo 3 necesario para realizar la virtualización inalámbrica, para establecer las comunicaciones se utiliza el protocolo B.A.T.M.A.N. sección A.4.1, donde se gestionan las comunicaciones, y es el servicio que permite el despliegue de aplicaciones sobre el sistema.

Una vez desplegados los agentes locales, el orquestador u orquestadores, se da el acceso al sistema operativo SOVORA mediante consola o mediante la interfaz web asociada a los servicios de monitoreo, el manejo de aplicaciones es descrito en la sección 7.10, ya que se debe implementar un modelo

particular para el despliegue de una aplicación distribuida sobre el sistema, esta distribución de aplicaciones permite generar políticas social inspiradas sobre le sistema, basadas en la cooperación y colaboración.

Finalmente se propone un modelo formal de la solución mediante el uso de bigrafos el cual fue descrito en el capítulo 4, basado en la investigación del profesor Robert Milner, se cierra con los test de prueba para validar el funcionamiento de los módulos propuestos y medir su rendimiento, recordemos que el presente trabajo tiene como objetivo mostrar la viabilidad de este sistema sobre una red ad hoc, adicionalmente de la posibilidad de implementar modelos sociales en ambientes distribuidos.

7.8. Modelo de la solución

Para desplegar el sistema en su totalidad se crearon dos elementos como se mostró en la sección anterior el Agente Local y el Orquestador, el diseño general es el mostrado en la figura 7-21, donde se ven las interacciones de los dos artefactos computacionales desarrollados, en la parte izquierda el agente local esta dotado con un módulo que permite dotar al nodo de características para operar en modo ad hoc, del mismo modo cuenta con un controlador que administra las aplicaciones controlando su throughput, estas aplicaciones están dentro de un contenedor de docker, el número de contenedores para este desarrollo es igual al número de núcleos disponibles en el dispositivo, en este caso se hace la gestión de los recursos de cómputo procesador y memoria, del mismo modo se realiza con los sensores o dispositivos de entrada y salida, el consumo de energía y el ancho de banda usados por el nodo, para ello se construyeron las imágenes necesarias para desplegar el sistema en diferentes arquitecturas.

Por su parte el orquestador establece el modo ad hoc de forma análoga al agente local, contiene los estados individuales del sistema y realiza la abstracción del estatus global de la solución, teniendo la información de todos los recursos administrables y la cantidad de recursos consumidos. El orquestador esta dotado con información de la caracterización de los dispositivos, esta caracterización es un servicio disponible en la solución para agregar dispositivos nuevos.

El pseudo código del Agente Local es el mostrado en el algoritmo 1 y el del orquestador es mostrado en el algoritmo 2, este modelo permite conocer los estados locales y aprovechar las ventajas de comunicación del modo ad hoc y la gestión de los contenedores. Otra característica adicional es la capacidad de desplegar aplicaciones desde el orquestador en cualquier nodo de la red.

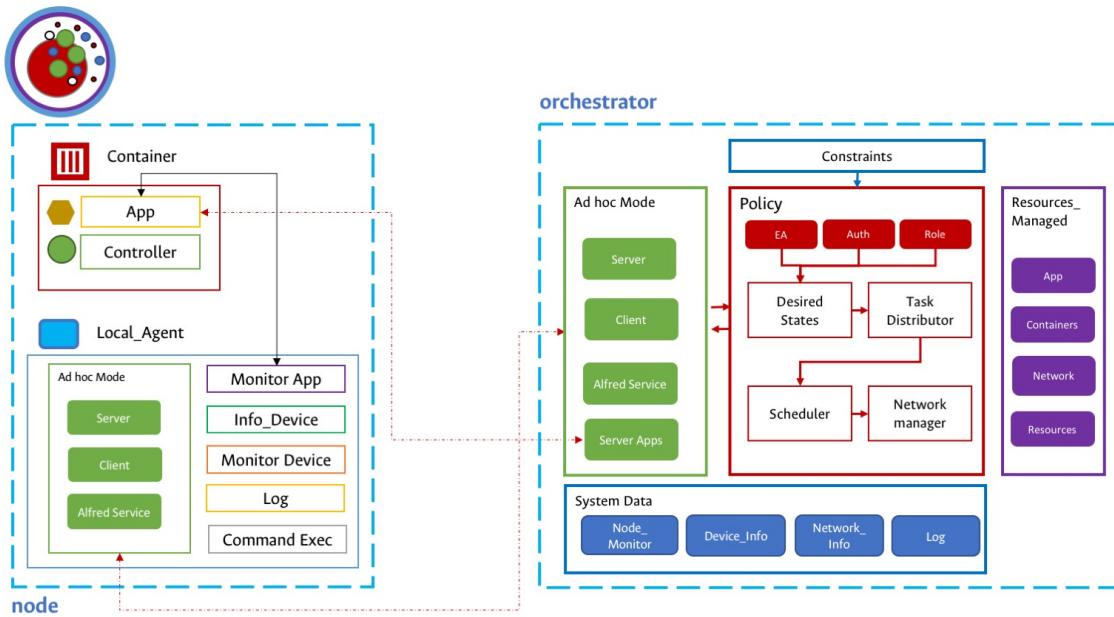


Figura 7-21: Orquestador y Agente Local

7.9. Modelo de Comunicación

Este esquema opera sobre redes inalámbricas en el modo ad hoc de las tarjetas de red, allí se activa el modulo batman-adv del kernel de linux para usar el protocolo B.A.T.M.A.N., este modelo permite la autoconfiguración de IP mediante el servicio avahi, de esta forma se hace una asignación dinámica de direcciones que es diseminada por el servicio de mensajería ALFRED, el orquestador tiene un mensaje etiquetado para que los nodos identifiquen ese rol y realicen los envíos de la información al nodo o nodos con esta etiqueta.

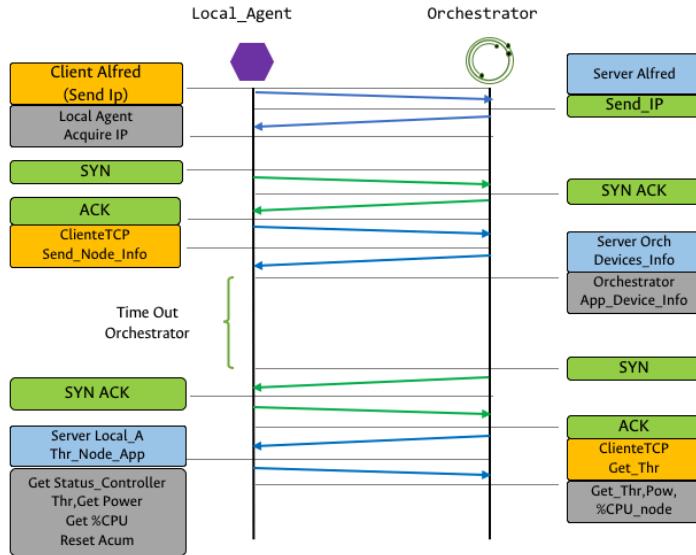


Figura 7-22: Modelo de comunicación

Como parte del desarrollo de la solución se creo un modelo de comunicación sobre la red con un conjunto de servidores y clientes UDP y TCP, multi-hilo en cada uno de los nodos, en sus diferentes roles, ya sea como Agente Local o como Orquestador. Esta comunicación envía por parte del agente local la información del estado actual y contiene un servidor activo para recibir los mensajes o tareas asignadas por el orquestador, del mismo modo el orquestador atiende las solicitudes del usuario, toma la decisión y hace el envío de la instrucción una vez realiza la consulta de estados globales, utiliza el módulo de toma de decisión y selecciona el nodo adecuado para asignar la tarea. Podemos dividir el esquema de comunicación en dos fases: la primer fase es la comunicación entre el orquestador y el agente Local la cual esta representada en la figura 7-22, allí se puede observar los clientes en amarillo, los servidores en azul, usados de forma concurrente para establecer la comunicación entre los diferentes miembros de la red, en gris están asociados los servicios que están operando con la información recopilada por los servidores de cada nodo miembros del sistema.

La segunda fase es la comunicación controlador-agente local, la cual se hace desde el agente local hacia el contenedor de Docker, donde se encuentran las aplicaciones y el controlador, que hace el enlace con las aplicaciones para monitorear el throughput, gracias a esta comunicación es posible también ajustar la velocidad de operación de las aplicaciones y en este sentido la cantidad de recursos utilizados por la aplicación, este esquema permite hacer monitoreo en tres niveles el primero el hardware, el segundo el estado de los recursos mapeados y el tercero las aplicaciones desplegadas sobre el sistema.

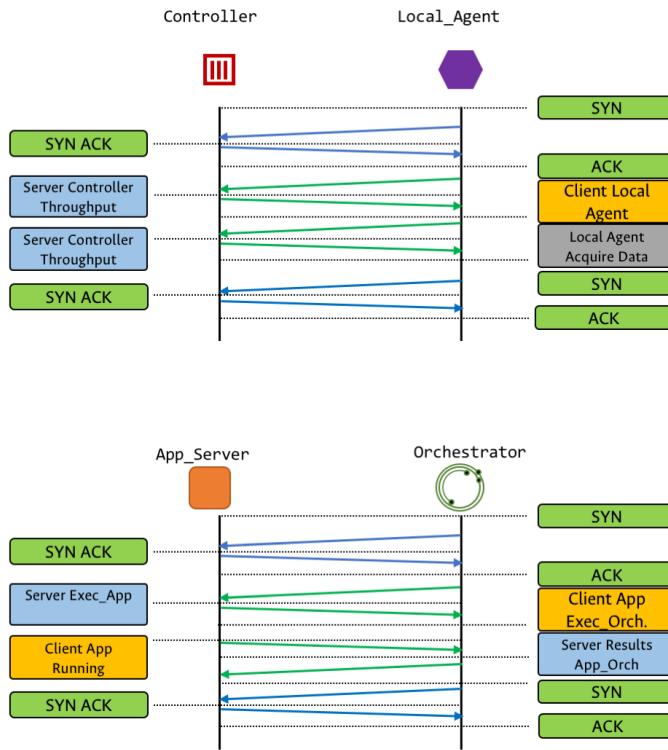


Figura 7-23: Modelo de comunicación Agente Local Aplicaciones

En el algoritmo 1 esta la rutina base del agente local y las tareas realizadas para mantener el sistema en operación y la comunicación entre nodos miembros del sistema y con el orquestador, en las líneas 1- 7 se inician los servicios de comunicación en modo ad hoc, al igual que el servicio de descubrimiento de orquestadores y los servidores de comunicación con las diferentes entidades señalizadas en el sistema. Posteriormente se despliegan las aplicaciones en contenedores de docker donde esta el controlador de aplicaciones y se procede a desplegar teniendo en cuenta el número de núcleos disponibles para su ejecución, en las líneas 12-36 esta el proceso principal donde tenemos arreglos donde se guarda el histórico de consumo de recursos en el nodo , a manera de ejemplo el recursos es el % CPU, esta tarea sea hace global a nivel de nodo y local a nivel de contenedor, del mismo modo se monitorea el throughput de las aplicaciones en ejecución, estos valores son almacenados y enviados al orquestador (líneas 37-39), este proceso, es importante resaltar la ejecución de políticas (líneas8-11), donde hay un hilo servidor que espera las instrucciones para ejecutar las políticas de ejecución de aplicaciones la cual esta contenida en el agente local.

El agente local desarrollado realiza dos tareas fundamentales, hacer la gestión de las comunicaciones en modo ad hoc mediante el protocolo BATMAN, el servicio de mensajería ALFRED y habilitar

Algorithm 1 Agente Local

Ensure: Communication Node Status

Require: Ad hoc Mode (B.A.T.M.A.N.) enable

```

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

```

las sesiones cliente servidor para los servicios de descubrimiento, estado de los nodos , ejecución de aplicaciones y monitoreo de recursos. Esta rutina esta contenida dentro de un *while*, cada vez que el orquestador envía una solicitud del estado del nodo, el agente local hace el cálculo de la media de los recursos, este proceso se hace con cada solicitud del orquestador, de esta manera se tiene el consumo de recursos en una ventana de tiempo determinada sin ser afectado por un registro histórico, este comportamiento le permite al orquestador tener información confiable a lo largo de la ejecución y conocer el estado real de los recursos distribuidos, es importante resaltar que el *delay* de esta comunicaciones es de 1 segundo aproximadamente,

Por su parte el orquestador cuenta con los mismos módulos de comunicación mostrados en el Agente Local, pero con diferencia al señalizar y presentarse como orquestador sobre la red, de esta manera los agentes locales autorizan el acceso a la información por parte del nodo con el rol Orquestador.

Al contener los estados globales el orquestador tiene la posibilidad de ejecutar políticas sobre el sistema, en dos sentidos, el primero apoyar las tareas de los nodos o usuarios sin recursos suficientes para ejecutar aplicaciones y el segundo distribuir los recursos para balancear el sistema y a carga de los miembros de la red.

Dentro de los recursos administrables por el orquestador están : cpu, memoria, disco, dispositivos de entrada y salida , ancho de banda, al igual que proveer información para las primitivas del sistema operativo SOVORA.

En el algoritmo 2 en las líneas 1 -5 se encuentran los servicios habilitados de Modo ad hoc y el servicio de ALFRED, el cual ha sido configurado como servicios de descubrimiento de nodos en el sistema, de esta forma se conocen las IP de todos los nodos dentro del sistema, este servicio es usado para que el nodo se presente con el rol de orquestador a los demás nodos (líneas 10-12), el ciclo principal (líneas 13-19) solicita a los m nodos presentes en el sistema la información del estado de los recursos y alimenta las tablas **7-26** de las políticas de tiempo de ejecución y las contrasta con las tablas de tiempo de diseño para ejecutar la política de balanceo de carga y control de los nodos sobre el sistema (Línea 26), la cual hace parte de la rutina de solicitud de servicios por parte del usuario (líneas 23-27).

De esta manera con estos dos artefactos se realiza el mapeo distribuido de los recursos y se obtiene la información general el sistema, la cual puede ser gestionada por el usuario con privilegios para gestionar los recursos, servicios y aplicaciones del sistema propuesto.

7.10. Modelo de Aplicaciones

Las aplicaciones desplegadas sobre este sistema también deben tener un modelo particular de ejecución, en este sentido las aplicaciones son desarrolladas bajo el concepto de virtualización ligera,

Algorithm 2 Orquestador

Ensure: Communication Network Status
Require: Ad hoc Mode (B.A.T.M.A.N.) enable
Require: Message Service (A.L.F.R.E.D.) enable

```

1: Run Log file
2: Run Server(STATES)
3: Run Server(EXEC)
4: Run Server(REGISTRY)
5: send(IPAddress) → ALFRED
6: while  $N \neq 0$  do
7:   set ServiceDiscoveryNode
8:   return (ip nodes)
9:   k=0
10:  if k % 5==0 then
11:    send(IP_Address) → ALFRED
12:  end if
13:  for m in nodes do
14:    client send(m,port,REQ")
15:    nodemcpu_acum ← cpu_acum_current
16:    nodemcont_acum ← cont_acum_current
17:    nodem Get Status(%CPU,Thr_App, Thr_Cont, Energy)
18:    nodem_info_node ← (Get Status,cpu_acum, cont_acum )
19:  end for
20:  save LocalLog
21:  save GlobalStatus ← Run TimeTables
22:  k++
23:  if User_Request == true then
24:    read(Design and Run Time Tables)
25:    run (DecisionModel)
26:    set(Policy) → (Node)
27:  end if
28: end while
```

bajo la arquitectura de microservicios visto en la sección 7.2, bajo esta arquitectura, las aplicaciones pueden estar dispersas en los diferentes miembros de la red y cuando son requeridas por algún nodo son recopiladas, esta distribución permite generar cooperación y un nuevo esquema de desarrollo de las aplicaciones.

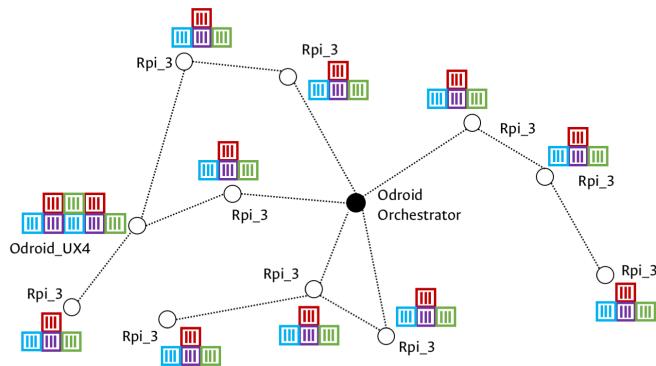


Figura 7-24: Modelos de Aplicaciones en Contenedores

En este desarrollo se propone usar aplicaciones ligeras para simular servicios locales sobre IoT, de este modo se usa el Benchmarking Suite *Mi Bench* Guthaus y cols. (2001), para tener una base de aplicaciones caracterizables y tener un nivel de referencia de su rendimiento y como pueden ser ajustadas para cambiar la velocidad de operación de la aplicación lo cual redundaría en la cantidad de recursos consumidos y por ende la energía. Estas aplicaciones se encuentran incluidas en contenedores, en este proceso se crearon las imágenes respectivas según la arquitectura del dispositivo y los servicios ofrecidos en el sistema, la distribución de aplicaciones y contenedores se puede ver en la figura 7-24, es importante indicar que el orquestador u orquestadores pueden disponer de la ejecución de estas aplicaciones, la herramienta usada para gestionar estos contenedores es Docker.

7.11. Política Desarrollada

Para la implementación de las políticas se seleccionó el modelo Observar -Decidir -Actuar, el cual se puede observar en la figura 7-25, esta política recibe como entrada datos del sistema sin procesar, en este caso el estado de los recursos tales como %cpu, estado de las aplicaciones, consumo de recursos de cada contenedor, energía, ancho de banda, hostname y tipo de tarjeta.

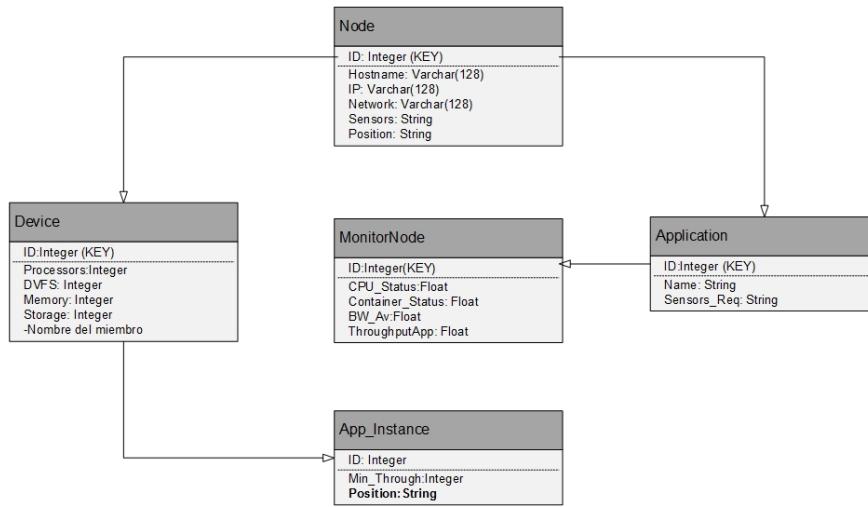


Figura 7-26: Tablas política Orquestador

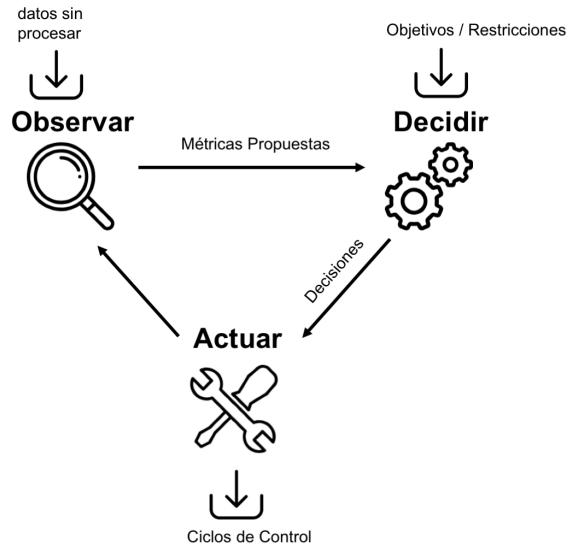


Figura 7-25: Política Observar - Decidir - Actuar

Esta información pasa a la segunda fase de decisión donde se validan las tablas generadas en tiempo de diseño y tiempo de ejecución estas se encuentran en el orquestador aunque pueden ser usadas por cualquier agente local, las tablas usadas por el orquestador (figura 7-26) se dividen en dos clases tablas en tiempo de diseño y tablas en tiempo de ejecución.

Tiempo de Diseño

- **Device Table:** Esta tabla contiene la información del dispositivo y características de la tarjeta en general , los campos de esta tabla son: número de procesadores , la capacidad de dvfs, información de memoria y almacenamiento del dispositivo.
- **ApplicationTable:** Esta tabla contiene la información de la aplicación a ser desplegada en cada dispositivo, los campos de esta tabla son: nombre de aplicación y los sensores requeridos y el máximo throughput alcanzado por tipo de dispositivo.

Tiempo de Ejecución

- **Node Table:** Esta tabla contiene la información de todos los nodos en la red con la información completa de recursos disponibles para desplegar aplicaciones, esta información es enviada por cada AL al orquestador, en un tiempo parametrizable en el orquestador, los campos de esta tabla son: el ID del nodo, tipo de dispositivo, hostname, la Ip, el SSID de red, los sensores y la posición del nodo. Esta tabla contiene información de dispositivos raspberry pi zero w, raspberry pi 2, raspberry pi 3 y odroid XU4.
- **Monitor Node Table:** Esta tabla recibe la información generada por el AL el monitoreo esencial de cada uno de los nodos asociando el ID del nodo y el tipo de tarjeta, la recepción de esta información también es parametrizable en el orquestador, esta tabla contiene información del monitoreo esencial para la toma de decisión, los campos de esta tabla son: ID del nodo, % de cpu consumida, % de Cpu por contenedor, Ancho de banda disponible, el valor de TQ del nodo (medido desde el orquestador a todos los nodos), las aplicaciones en ejecución junto con su throughput.
- **App Instance Table:** Esta tabla contiene la información de la aplicación en ejecución los campos de esta tabla son: ID del nodo donde se ejecuta, throughput mínimo, throughput máximo y posición del nodo donde se ejecuta la aplicación.

Los campos claves en la distribución de carga y selección del nodo son el throughput deseado para la ejecución de la aplicación y el TQ de los nodos candidatos, este proceso realiza una búsqueda en las tablas de dispositivos, posteriormente en la de aplicaciones para definir los nodos candidatos que ofrecen el throughput deseado, finalmente valida el estado del nodo con el histórico de la tabla de monitoreo para enviar la solicitud de ejecución al AL.

La política seleccionada por el orquestador es enviada como un mensaje de ejecución al AL, con la información de la aplicación a ejecutar, entregando la información correspondiente para el ajuste de la cuota de recursos, el contenedor a desplegar y el destino, esto permite conocer la ubicación exacta tanto lógica como física de la aplicación en ejecución. Existen dos políticas desplegadas en el sistema

la política de distribución de carga y la política de gestión de recursos, la primera se despliega en el orquestador y la segunda en el agente local como ejemplo se tiene el caso particular del dispositivo Odroid XU4, el cual permite el control de DVFS esta política se ve en el algoritmo 3.

En el algoritmo 3 la primera fase *Observar* descrita en las líneas 2-4, obtiene el estatus de las aplicaciones y recursos consumidos, en este caso la %cpu, como valores de entrada o datos sin procesar, en la segunda etapa *Decidir* comprendida en las líneas 6 a 33 se realiza la validación del throughput deseado y el máximo throughput entregado por el sistema, el objetivo de esta fase es conocer la frecuencia de operación del dispositivo y el throughput asociado para ajustar el valor de la frecuencia que contribuya a la disminución de energía, la frecuencia puede ser escalada si se encuentra al máximo de cpu y el dispositivo lo permite de lo contrario no se escala y la aplicación es detenida, en este caso la aplicación debe ser lanzada en otro nodo, de forma analoga se busca un valor de retardo de las aplicaciones para disminuir la cantidad de knobs o ciclos de ejecución de la aplicación en el contenedor. El valor de retardo de las aplicaciones es calculado con el valor de throughput deseado y el throughput actual, esta rutina esta en las líneas 9-16 donde se valida la necesidad de escalar la frecuencia solo si la frecuencia actual es menor a la frecuencia máxima del dispositivo. Al escalar la frecuencia se busca la relación voltaje/frecuencia para encontrar el throughput deseado y el nivel de energía mínimo para ese fin, esta etapa se realiza en las líneas 23 -33 donde se incrementa gradualmente la frecuencia para reiniciar el proceso de muestreo, en la fase final actuar el valor de obtenido de la frecuencia adecuada f_{new} es indicado al sistema operativo ajustando el valor de frecuencia mientras a la aplicación se le genera un retardo $a.q_{new}$ el cual es ajustado por el controlador.

Algorithm 3 Política de gestión de recursos AL

```

1: {FASE OBSERVAR}
2:  $appls \leftarrow get\_appls\_status()$ 
3:  $U \leftarrow get\_cores\_utilization()$ 
4:  $f_{curr} \leftarrow get\_curr\_freq()$ 
5: {FASE DECIDIR}
6:  $appls_{kill} \leftarrow \emptyset$ 
7:  $upscale \leftarrow false$ 
8: for  $a \in appls$  do
9:    $a.q_{new} \leftarrow a.q \cdot \frac{a.th_{target}}{a.th_{curr}}$ 
10:  if  $a.q_{new} > 100\%$  then
11:    if  $f_{curr} < f_{max}$  then
12:      if  $upscale = false$  or  $q_{max} < a.q_{new}$  then
13:         $q_{max} \leftarrow a.q_{new}$ 
14:      end if
15:       $upscale \leftarrow true$ 
16:       $a.q_{new} \leftarrow 100\%$ 
17:    else
18:       $appls_{kill} \leftarrow appls_{kill} \cup \{a\}$ 
19:       $appls \leftarrow appls / \{a\}$ 
20:    end if
21:  end if
22: end for
23: if  $period_{DVFS} = true$  then
24:   if  $appls \neq \emptyset$  then
25:      $f_{new} \leftarrow f_{min}$ 
26:   else
27:     if  $upscale = true$  then
28:        $q_{max} \leftarrow max(U)$ 
29:     end if
30:      $f_{new} \leftarrow f_{curr} \cdot \frac{q_{max}}{100\%}$ 
31:      $f_{new} \leftarrow \left\lfloor \frac{f_{new}}{f_{step}} \right\rfloor \cdot f_{step} + f_{step}$ 
32:   end if
33: end if
34: {FASE ACTUAR}
35: for  $a \in appls_{kill}$  do
36:   kill  $a$ 
37:   notify_fo( $a$ )
38: end for
39: if  $period_{DVFS} = true$  then
40:   actuate  $f_{new}$ 
41: else
42:   for  $a \in appls$  do
43:     actuate  $a.q_{new}$ 
44:   end for
45: end if

```

Capítulo 8

Pruebas y Resultados

Para el desarrollo de este trabajo se empleó un conjunto de dispositivos embebidos heterogéneos, en su mayoría Raspberry Pi (modelos 3, 2, zero w) y odroid (XU4), interfaces inalámbricas IEEE 802.11/g/b/c para activar el modo Ad hoc en algunos dispositivos con limitaciones de comunicación. Para las mediciones de energía en todos los dispositivos se adquirió el smartpower meter 2.

Sobre estos dispositivos se usaron distribuciones de sistemas operativos libres y trabajando con la versión de kernel 4.9.41-v7+, python 2.7 para la programación de los módulos agente local, orquestador, el uso de Docker v 17.09.1-ce, para el despliegue de las aplicaciones del benchmarking suite MiBench, las cuales están desarrolladas en c++, se realizaron las imágenes para arquitectura armv7 y armv5. Se procedió con la caracterización de dispositivos para el mapeo de recursos y conocer los estados de operación de los dispositivos adquiridos.

8.1. Caracterización de los Dispositivos

Para realizar las primeras pruebas de los diseños propuestos en este trabajo, se realizó una caracterización de los dispositivos tanto en cómputo (memoria, cpu, almacenamiento y red), como del comportamiento del protocolo B.A.T.M.A.N. para entender su funcionamiento, consumo de recursos y forma de operación sobre los dispositivos. Finalmente comprender cual es el estado óptimo de operación y las limitaciones de los dispositivos como de las aplicaciones y elementos escogidos durante el tiempo de ejecución. En la tabla 8-1 se ven los dispositivos usados arquitectura y demás características.

Para caracterizar los dispositivos se usó como MiBench Multimedia BenchMarking suites para ver el consumo de recursos y el tiempo de ejecución con algoritmos estandarizados. Estos algoritmos son de

device	Architecture	Processor Fam.	Chip	Cores	arm_fre(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

Tabla 8-1: Arquitectura de los dispositivos usados

Aplicación	Algoritmo	Recursos Consumidos
Seguridad	SHA	CPU, Memoria, Ancho de banda
Procesamiento de Imagen	JPEG	CPU, Memoria, Ancho de Banda
Análisis de Datos	CRC	CPU, Memoria, Ancho de banda

Tabla 8-2: Aplicaciones seleccionadas para la realización de las pruebas

consumo constantes de cpu, es decir ocupan el 100 % de los recursos de cómputo al ser ejecutados, sirven para caracterizar el consumo de energía del dispositivo y la tasa de ocupación de recursos, tiempo y rendimiento de la aplicación según la arquitectura del dispositivo

Para el desarrollo de estas pruebas se utilizaron aplicaciones potenciales en el ecosistema de IoT como los son el cifrado de información con el algoritmo *sha*, el procesamiento de imágenes con los algoritmos *jpeg* y *susan*, y la codificación de audio con el filtro *adcpm*, estas aplicaciones fueron modificadas para obtener primero información del throughput de las aplicaciones enlazadas con el controlador de aplicaciones el cual permite controlar la velocidad de ejecución de las aplicaciones, este elemento permite enlazar la operación de las aplicaciones con las políticas del sistema, del mismo modo permite enviar el resultado del algoritmo a la ip indicada al momento de sus ejecución, ya sea un orquestador o un nodo de la red. El resumen de las aplicaciones usadas se puede ver en la tabla 8-2.

Este test fue realizado en las tarjetas descritas en la tabla 8-1, los resultados de estas pruebas validan el consumo de energía y de cpu al 100 % este primer procedimiento se ve en las tablas 8-3,8-4,8-5 para el algoritmo *crc32*, en las tablas 8-6, 8-7, 8-8 para el algoritmo *sha*, en las tablas 8-9,8-10 y 8-8 para la codificación *jpeg*, estos datos se convierten en los insumos para las políticas desplegadas por el orquestador, los resultados de estas tablas son la media de 100 experimentos realizados con el benchmarking suite.

	<i>RPi_3CRC32</i>	26MB	260M	1G
%CPU	98	100	100	
Tiempo (s)	0.1	4.234	12.34	
Potencia (mW)	1527.96	1530.16	1517.43	

Tabla 8-3: Resultados algoritmo CRC32 - MiBench RPi 3

Esta tablas evidencian un consumo cercano a 1520 mW de consumo al ser ejecutadas las aplicaciones con diferentes entradas y con una cantidad de repeticiones, cada experimento se realizó cien veces y el producto final es la media de estas mediciones. Estas primeras mediciones se realizaron con la antena inalámbrica apagada.

	<i>RPi_zeroCRC32</i>	26M	260M	1G
%CPU	82.5	94	94	
Tiempo (s)	0.2	6.23	15.7	
Potencia (mW)	652.32	651.3		

Tabla 8-4: Resultados algoritmo CRC32 MiBench Rpi zero

	<i>OdroidCRC32</i>	26MB	260MB	1GB
%CPU	100	100	100	
Tiempo (s)	0.01	0.10	12	
Potencia (mW)	4401.724	444.78	4390.9	

Tabla 8-5: Resultados algoritmo CRC32 MiBench Odroid

Estos resultados muestran el consumo de energía promedio de una cpu al 100 % en los tres dispositivos usados, de este modo se puede calcular la media de consumo de energía y conocer el consumo máximo de energía en un dispositivo físico. Las tablas 8-6, 8-7 y 8-8 muestran los resultados de recursos consumidos en el algoritmo *sha*.

	<i>RPi_3SHA</i>	300KB	30MB	300MB
%CPU	100	100	100	
Tiempo(s)	0.103	1.14	10.671	
Potencia(mW)	1564	1573.25	1591.4	

Tabla 8-6: Resultados algoritmo SHA MiBench RPi 3

<i>RPi_zeroSHA</i>	300 KB	30 MB	300 MB
%CPU	90	94	94
Tiempo (s)	0.187	1.89	14.135
Potencia (mW)	651.15	651.31	651.22

Tabla 8-7: Resultados algoritmo SHA MiBench RPi zero

<i>OdroidSHA</i>	300 KB	30 MB	300 MB
%CPU	100	100	100
Tiempo (s)	0.036	0.029	2.87
Potencia (mW)	6597	6654	6590.44

Tabla 8-8: Resultados algoritmo SHA MiBench Odroid

De forma análoga se realizó el proceso con la aplicación jpeg, para la codificación de imágenes, este proceso, es usado en el contexto de servicios de vídeo sobre la red, como la consecución de imágenes, adicionando tráfico sobre el sistema, estas mediciones son realizadas en ciclos de codificación cien veces por cada entrada, los resultados se pueden observar en las tablas **8-9, 8-10 y 8-11**.

<i>RPi_3jpeg - dec</i>	512x512 px	1024x1024 px	2048x2048 px	4096x4096 px
%CPU	100	100	98	100
Tiempo (s)	0.121	0.323	2.263	10.714
Potencia(mW)	1519.91	1535.21	1531.84	1537.42

Tabla 8-9: Resultados algoritmo Jpeg MiBench RPi 3

<i>RPi_zerojpeg - dec</i>	512x512 px	1024x1024 px	2048x2048 px	4096x4096 px
%CPU	94	94	94	94
Tiempo (s)	0.10	1.46	2.16	14.91
Potencia (mW)	651.15	651.21	651.34	

Tabla 8-10: Resultados algoritmo Jpeg MiBench RPi zero

<i>odroid - jpeg - dec</i>	512x512 px	1024x1024 px	2048x2048 px	4096x4096 px
%CPU	100	100	100	100
Tiempo (s)	0.002	0.0085	0.59	0.413
Potencia (mW)	4020.2	4120	4108.99	4842

Tabla 8-11: Resultados algoritmo Jpeg MiBench odroid

8.1.1. Comunicación modo Ad hoc

Para el modo ad hoc se realizó el test batctl tcpdump en la versión de B.A.T.M.A.N. 2017-2 , esta prueba consiste en enviar 14 MB entre dos nodos ad hoc, sobre la interfaz bat0 (Interfaz wlan en modo ad hoc). Esta es una de las opciones de prueba del protocolo , con esta prueba se caracterizó el consumo de energía al transmitir información ,al igual que tener una base del rendimiento de las comunicaciones entre los nodos bajo este protocolo, los resultados se pueden ver en las tablas 8-12, 8-13 y 8-14

<i>RPi_3batctl</i>	14 MB rpi_zero	14 MB Odroid
%CPU	100	100
Tiempo (s)	10.19	10.54
Potencia (mW)	2211.88	2230
Energía (J)	2211.88	2230
Throughput (MB/s)	0.8893	1.7
MB Enviados	7.605	18.5

Tabla 8-12: Resultados prueba Ad hoc batctl/ batman-adv RPi 3

Se puede evidenciar un aumento en el consumo de energía respecto a las tablas presentadas en la sección anterior, este hace que este tipo de sistemas tenga como base de operación el consumo de energía y sea una de las posibles políticas implementadas en el sistema.

<i>RPi_zero batctl</i>	14 MB rapi_3	14 M Odroid
%CPU	94	94
Tiempo (s)	10.233	10.05
Potencia (mW)	1052.23	1051.4
Energía	1052.23	1051.4
Throughput (MB/s)	13.11	10.3
MB Enviados	13.11	10.3

Tabla 8-13: Resultados prueba Ad hoc batctl/ batman-adv RPi zero

Como se evidencia en las tablas de esta sección existen variaciones en el throughput de los test debido a las condiciones del medio donde se toma la prueba y el ruido en el área de pruebas, estos experimentos se realizaron cien veces y el resultado es su media.

8.1.2. Caracterización Modelo de Energía

Para gestionar la energía como recurso dentro del sistema propuesto, se realizó una caracterización de los consumos de energía y los costos de transmisión de paquetes dentro del sistema en modo ad hoc, de esta manera se obtuvo un modelo de consumo, elemento base para la selección de políticas dentro del orquestador.

	<i>Odroidbatctl</i>	14 MB rpi_zero	14 MB rpi_3
%CPU	56	57	
Tiempo (s)	10.224	10.1	
Potencia (mW)	4429.34	4430	
Energía	3997.95	4008	
Throughput (MB/s)	1.38	1.38	
MB Enviados	14.5	14.6	

Tabla 8-14: Resultados prueba Ad hoc batctl/ batman-adv Odroid

El procedimiento de caracterización de los consumos de energía fueron hechos tomando medidas en estados idle (libre), idle con wifi encendido, active (consumos cpu 100 %) y active con wifi, estas medidas fueron realizadas con el Smart Power Meter 2, esta herramienta mantiene una entrada constante de voltaje y una corriente estable para hacer las mediciones, del mismo modo se hicieron medidas con throughput de 100 %, 75 % y 50 % sobre los dispositivos para hacer una interpolación de estos datos, para obtener una función que describa el consumo de energía del dispositivo elemento clave en las políticas y el mantenimiento de la red en el tiempo.

Las aplicaciones son ejecutadas dentro de contenedores como sistemas aislados controlados, se usó la interfaz de c-groups para asignar recursos dentro del sistema. Los resultados de esta caracterización se pueden ver en las tablas **8-15, 8-16, 8-17, 8-18 y 8-19**

LITTLE 600 MHz ODROID				
APP	INPUT	THR_MAX	75 %THR_MAX	50 %THR_MAX
SHA	INPUT_LARGE	6	4,5	3
SUSAN	INPUT_LARGE	1,5	1,125	0,5
ADPCM	INPUT_LARGE	12	9	7
JPEG	20.RGB	0,5	0,5	0,5
KMEANS	20.RGB	0,5	0,5	0,5

Tabla 8-15: Throughput ODROID Frecuencia Little 600 MHz XU-4

Para el Odroid es importante resaltar la existencia de ocho núcleos cuatro de arquitectura big y cuatro de arquitectura little , los cuales pueden variar su frecuencia para operar sobre este sistema se caracterizaron diversas frecuencias de operación de la misma forma que la tabla **8-15**. Las demás mediciones se pueden ver en el anexo E.

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

Tabla 8-16: Consumo de Energía RPi 3 con Wifi

POWER NO_WIFI RPi 3				
	IDLE	50 %THR_MIN	75 %THR_MAX	THR_MAX
APP	1535,92	1671,31	1771,11	1808,42
SHA	1535,92	1669,59	1724,61	1806,59
ADPCM	1535,92	1668,22	1732,01	1716,68
SUSAN	1535,92	1671,33	1718,28	1749,7
JPEG	1535,92	1671,33	1718,28	1749,7

Tabla 8-17: Consumo de Energía RPi 3 sin Wifi

Estos valores son producto de muestreos realizados con las aplicaciones ejecutandose inicialmente al throughput máximo (cpu 100 %) y luego ajustado por el controlador de aplicaciones dentro del contenedor, este modelo permite ajustar la velocidad de las aplicaciones y de esta manera controlar la energía esta es una de las propiedades expuestas en el orquestador diseñado.

POWER+ WIFI RPi Zero			
	50 %THR_MIN	75 %THR_MAX	THR_MAX
APP	848,3	912,172	1047,9
SHA	848,3	894,707	998
ADPCM	848,3	869,75	1002,99
SUSAN	848,3	869,75	998
JPEG	848,3	869,75	998

Tabla 8-18: Consumo de Energía RPi zero con Wifi

POWER NO_WIFI RPi Zero			
	50 %THR_MIN	75 %THR_MAX	THR_MAX
APP	761,805	798,4	913,17
SHA	761,805	798,4	913,17
ADPCM	761,805	795,2	888,22
SUSAN	761,805	795,2	898,2
JPEG	761,805	795,2	898,2

Tabla 8-19: Consumo de Energía RPi zero sin wifi

Los resultados obtenidos de interpolar y validar los datos medidos están representados en las figuras 8-1, 8-2, y 8-3, con base en todos los experimentos desarrollados.

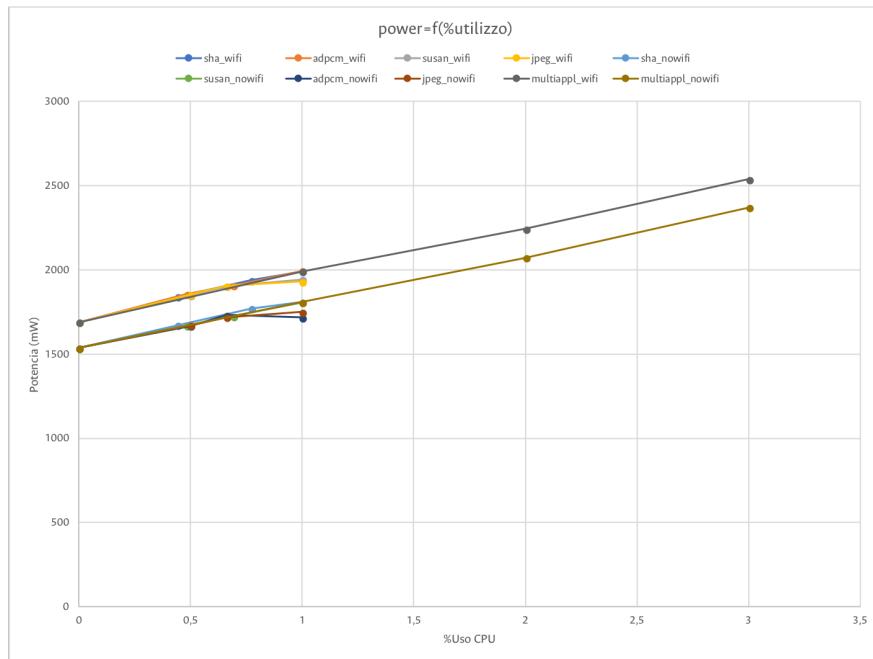


Figura 8-1: Modelo de Energía RPI 3

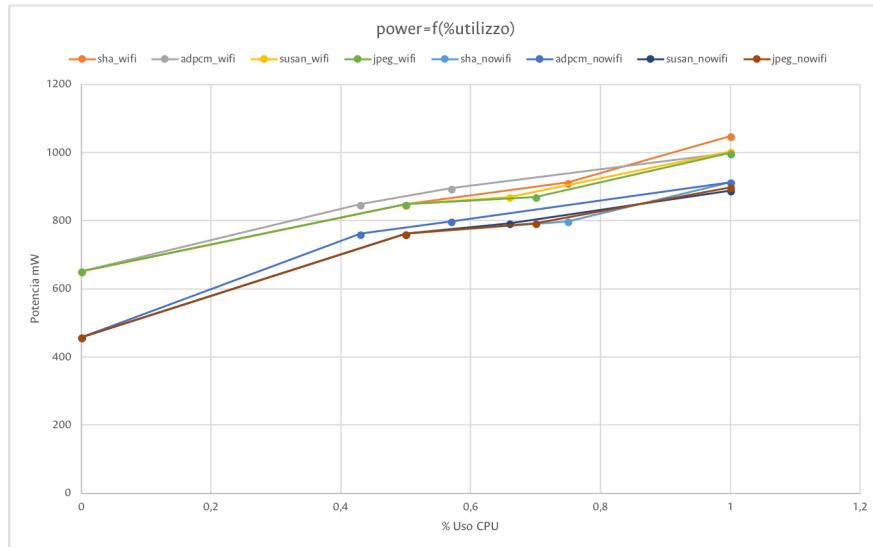


Figura 8-2: Modelo de Energía RPI zero

$$P_{\%Util} = P_{idle} + (P_{100\%} - P_{idle}) * \%Util \quad (8-1)$$

Podemos concluir con base en esta experimentación que el consumo de energía en dispositivos rpi (Modelo 3 y zero) se puede modelar empleando la ecuación 8-1. Para el dispositivo Odroid se reali-

zó el mismo procedimiento, adicionalmente se validó el modelo con cambios de frecuencia, en los diferentes core Big y Little, el siguiente es el modelo propuesto para un odroid en la ecuación 8-2.

$$\begin{aligned}
 P(freq_little, freq_big, util_little\%, util_big\%) = & \\
 & P_idle_min_freq \\
 & + D_P_idle_little(freq_little) \\
 & + D_P_work_little(freq_little) * \% \\
 & + util_little D_P_idle_big(freq_big) \\
 & + D_P_work_big(freq_big) * \%util_big
 \end{aligned} \tag{8-2}$$

Donde:

- $P_idle_min_freq$ es la potencia en reposo a la frecuencia mínima (400 MHz en nuestro caso)
- $D_P_idle_little(freq_little)$ es el aumento de potencia en condiciones de ralentización para una frecuencia dada la frecuencia más pequeña en relación con $P_idle_min_freq$. Se obtuvieron los promedios de las diferencias entre la potencia y la frecuencia mínima $freq_little - P_idle_min_freq$.
- $D_P_idle_big(freq_big)$ se calcula como el valor anterior.
- $D_P_work_little(freq_little)$ se calculó para cada frecuencia restando la potencia en intervalos pequeños de $\%util = 1$, es decir el 100 % de consumo de *cpu* y de frecuencia $(+P_idle_min_freq D_P_idle_little(freq_little))$
- $D_P_work_big(freq_big)$ se calculó del mismo modo que es en el punto anterior.

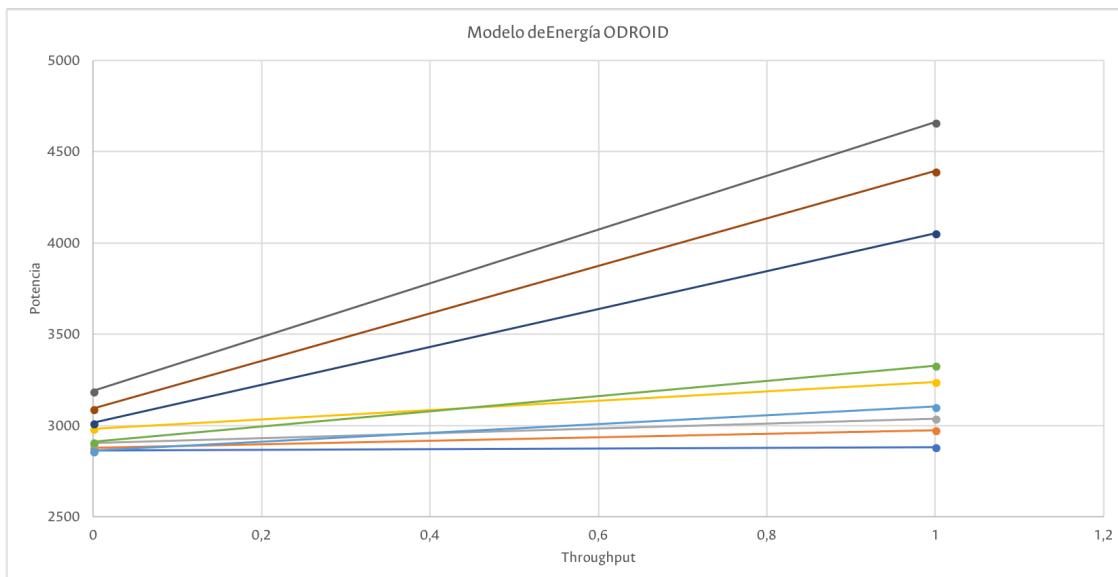


Figura 8-3: Modelo de Energía Odroid XU 4

8.2. Comparación Cualitativa de la Solución Propuesta

Para realizar una validación cualitativa de la solución SOVORA realizada es importante indicar que los sistemas operativos distribuidos inalámbricos, no se encuentra fácilmente en la literatura pero si algunas soluciones comerciales en el mercado, como las mostradas en la tabla 8.2, la cual muestra los elementos usados en y desarrollados en este trabajo, en comparación con las soluciones comerciales actuales.

Dado las características de los sistemas operativos distribuidos mencionados en el presente trabajo se han seleccionado los siguientes componentes para su valoración:

1. Agente Local: Algunas tecnologías no usan elementos invasivos o con capacidades de monitorear, manipular e instanciar recursos o código sobre el sistema, sin embargo existen agentes locales que permiten hacer esta tarea con restricciones para evitar fallas o atentar contra la integridad del sistema.
2. Medio de Transmisión: La mayoría de las soluciones han sido desarrolladas para ser desplegadas sobre enlaces alámbricos cobres o fibras para un alto desempeño, sin embargo la solución propuesta en este trabajo puede operar en ambientes inalámbricos, lo cual lo hace útil en ambientes característicos de sistemas IoT.

3. Tipos de Virtualización y Orquestación: En estos ítems podemos comparar la existencia de estos artefactos en soluciones comerciales pero con una cantidad mayor de restricciones, en cuanto al despliegue de políticas, gestión de recursos , modificación del código y adición de funciones. El orquestador es una capa de control adicional que se ha incluido en estos proyectos dada la alta escalabilidad de los sistemas y las necesidades de control de dispositivos dentro de los ambientes IoT.
4. Monitoreo: Es una de las características esenciales de cualquier sistema de cómputo, aún más en estos ambientes hostiles como los inalámbricos y las restricciones de recursos de los dispositivos de borde en el ecosistema IoT. Por ello la solución propuesta no se limita únicamente a monitorear los recursos sino también las aplicaciones contenidas en los contenedores o desplegadas en el sistema operativo distribuido.
5. Seguridad: es uno de los componentes principales en el diseño de un producto de cómputo, este elemento en las aplicaciones comerciales ha sido incluido paulatinamente, en el caso de este proyecto hay una protección inicial tomada del kernel de Linux y de los permisos de los artefactos desarrollados, sin embargo las comunicaciones pueden ser mejoradas, cifradas y protegidas con diferentes técnicas de autenticación, autorización y seguimiento de las acciones hechas por un usuario.
6. Interoperabilidad y escalabilidad: En cuanto a estos ítems el primero es clave para la escalabilidad tanto horizontal como vertical del sistema desarrollado, la mayoría de soluciones del mercado están abriendo y compartiendo sus protocolos para garantizar la interoperabilidad de las soluciones. Adicionalmente la elasticidad en condiciones dinámicas el cambio de los recursos de cómputo disponibles para ejecutar aplicaciones es una característica base para el diseño de sistemas en ambientes inalámbricos , dinámicos y estocásticos.

Tabla 8-20: Comparación Cualitativa con otras soluciones similares en el mercado

Fabricante	Nombre de la Solución	Medio de Transmisión	Tipo de Virtualización	Nombre del Orquestador	Edición de Políticas	Agente Local	Mon. Recursos	Mon. de Apps	Seguridad	Interoperabilidad / Escalabilidad
Vmware	Vmware Vsphere	Alámbrico	PCI - Docker	Kubernetes; Docker Swarm	NO	SI	NO	SI	SI	SI
Oracle	Oracle VM	Alámbrico	Oracle Cloud Infrastructure - Docker	Kubernetes	NO	NO	SI	SI	SI	SI
Amazon	AWS	Alámbrico	AWS - Docker	Amazon EC2	NO	NO	SI	PARCIAL	SI	SI
Red Hat	Red Hat Virtualization	Alámbrico	OpenShift	OpenShift	NO	NO	SI	NO	SI	SI
Huawei	Fusion Sphere	Alámbrico	Cloud Container Engine - Docker	Kubernetes	NO	SI	SI	NO	SI	SI
SistechTQIN	Sovra	Inalámbrico	Orquestador TQIN	Orquestador TQIN	SI	SI	SI	SI	NO	SI
Microsoft	Azure	Alámbrico	Azure Container Service	Kubernetes; Docker Swarm - Marathon	NO	NO	SI	PARCIAL	SI	SI
Apache Mesos	Marathon	Alámbrico	Marathon	Kubernetes; Docker Swarm	NO	NO	SI	NO	SI	SI
Hasdicorp	Nomad	Alámbrico	Nomad	Consul	SI	NO	SI	NO	SI	SI

8.3. Resultados de Operación de la Solución

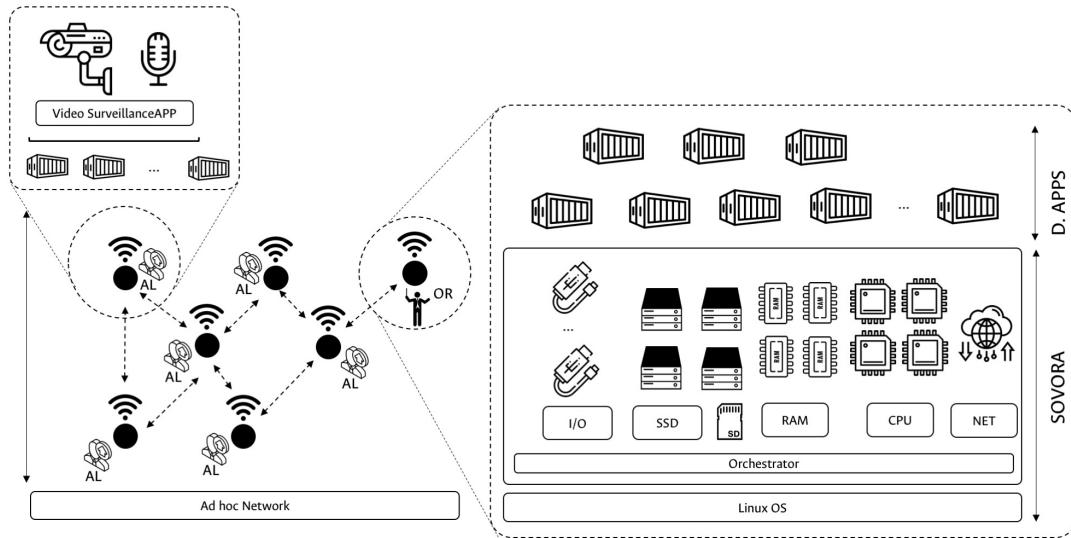


Figura 8-4: Escenario de Pruebas

Para el desarrollo del modelo de operación de S.O.V.O.R.A. se seleccionó como escenario de prueba un servicio de vídeo vigilancia que opera sobre el sistema como se ve en la figura 8-4, cada una de las aplicaciones están desplegadas sobre contenedores de docker bajo la arquitectura de micro servicios, estas aplicaciones están distribuidas sobre los nodos presentes en la red ad hoc, el sistema operativo virtualizado toma la información de los recursos mapeados y alimenta las tablas del estatus del sistema, esto permite distribuir tareas dependiendo de la política de control programada y del objetivo del sistema. A manera de ejemplo se muestran estas interacciones entre los nodos, la red y las unidades básicas desarrolladas en este sistema el agente local y el orquestador.

Del mismo modo los modelos social inspirados realizados están basados en las formas de estado y gobierno usados dentro del esquema de pruebas son los indicados en la sección 5.1.5, son: el modelo de PseudoEstado Monarquía: Absoluta y los modelos de República Parlamentaria y Presidencialista.

La monarquía absoluta tiene como eje al Orquestador quien dirige todas las tareas del sistema y toma las decisiones que considere pertinentes para la correcta operación del sistema. En el caso de República Presidencialista inicialmente hay un consenso donde se decide quien toma el rol de Orquestador y este puede ser cambiado periódicamente dependiendo de las dinámicas del sistema, por último la República Parlamentaria todos los nodos en consenso toman las decisiones en el sistema y realizan las operaciones de distribución de carga y tareas dentro del sistema.

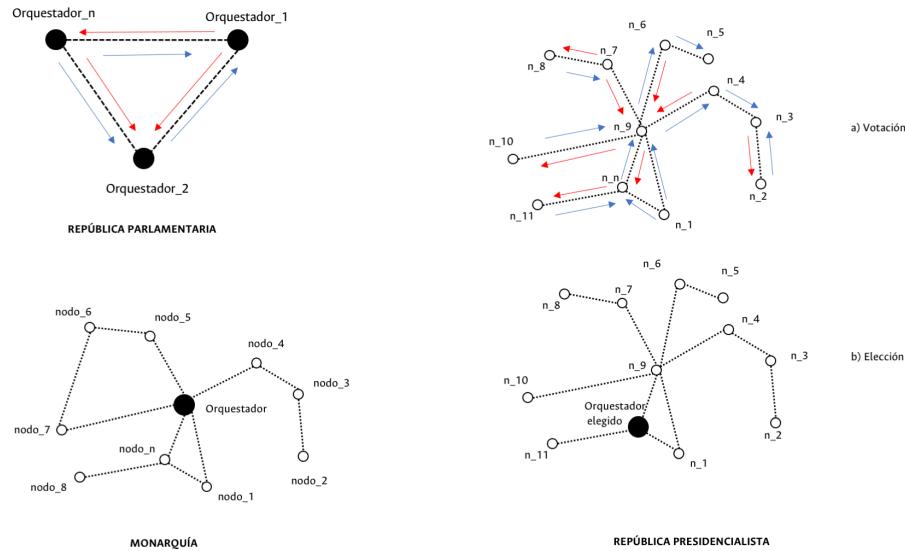


Figura 8-5: Modelos de Pseudo Estado implementados

El primer modelo es el de *Monarquía* donde un nodo tiene el rol de orquestador desde el inicio de la red y hace la gestión del sistema, según las decisiones tomadas por este nodo, el segundo escenario presenta una votación inicial, esta es la *República Presidencialista*, donde se elige el nodo que realizará el rol de Orquestador y hará la gestión del sistema, este proceso de votación puede ser configurado para ser repetido en un intervalo de tiempo específico, finalmente esta el modelo más complejo donde todos los nodos están bajo el rol de orquestador y todos deciden la acción subsecuente, de esta forma se instaura la *República Parlamentaria*.

Los modelos propuestos están diagramados en la figura 8-5.

Las interacciones entre los nodos se hace de manera directa sobre el canal inalámbrico con la combinación del servicio ALFRED y B.A.T.M.A.N. para hacer los procesos de votación y consenso, haciendo un poco más ligera la carga de información sobre la red, estos tres modelos son evaluados en detalle como se muestra en la tabla 8-21.

Item	Escenario 1	Escenario 2	Escenario 3
Movilidad		x	x
Aplicaciones Distribuidas	x	x	x
Múltiples Orquestadores			x
Múltiples Agentes Locales		x	
Políticas de Consumo de Energía	x	x	x
Políticas de Consumo de Recursos	x	x	x
Algoritmo de Consenso		x	x
Docker	x	x	x
Despliegue de Instituciones		x	x

Tabla 8-21: Escenarios Validados

Las siguientes son las operaciones realizadas por el agente local y el orquestador.

Operaciones de las abstracciones de S.O.V.O.R.A.

- **Agente Local sobre el Sistema Operativo:** La tareas realizadas sobre el sistema operativo son la medición del consumo cpu, medición del consumo de memoria RAM, medición del throughput de las aplicaciones en los contenedores , gestión del espacio de almacenamiento asignado y mapeo de dispositivos de entrada salida (bajo este esquema otras tareas pueden ser añadidas como parte de las rutinas del Agente Local), en particular el consumo de CPU medido en porcentaje es el valor de entrada para calcular el consumo de energía usando el modelo propuesto en la sección 8.1.2

Las mediciones de energía del sistema son realizadas a través de llamadas al sistema operativo del nodo, a través de la consulta del consumo de CPU del dispositivo, llamadas por núcleo del dispositivo y presentadas en mW; por cada contenedor desplegado se realiza el mismo proceso para conocer su consumo de CPU al ejecutar las aplicaciones en los mismos. Es importante recordar que la imagen de Docker desarrollada para este proyecto incluye el controlador, las aplicaciones y los módulos de comunicación entre los contenedores y el agente Local. Del mismo modo se hace el monitoreo de la memoria disponible y el recurso reservado de espacio en disco. El agente local tiene puertos como medio de comunicación para ejecutar comandos y la realización del consenso en los diferentes modos de operación.
- **Agente Local sobre el Contenedor :** El agente local tiene la capacidad de desplegar contenedores, migrar contenedores, detener aplicaciones desplegadas en contenedores, y conocer el estatus de las aplicaciones ejecutadas en los contenedores. Mediante un sistema de comunicación por puertos recibe las ordenes del orquestador para ajustar el throughput de las aplicaciones de acuerdo a las políticas desplegadas en el sistema.
- **Orquestador:** Este artefacto computacional es otro tipo de agente sobre el sistema con mayor jerarquía sobre el sistema, tiene la capacidad de indicar los comandos a realizar por los Agentes Locales de acuerdo con las políticas del sistema, contiene toda la información del sistema en tablas, (figura 8-24) que le permiten conocer la arquitectura del dispositivo, el estado actual del dispositivo, el máximo throughput de las aplicaciones en un dispositivo y el estado actual de la aplicación. Esta solicitud de información es parámetrizable y constituye el monitoreo esencial de este sistema, adicional contiene la consola de operación base y la interfaz web para el despliegue de información del estado de los nodos.

Como resultado se puede visualizar el estado del nodo en una interfaz web de SOVORA como se ve en la figura 8-6, o manipular desde la consola de comandos la cual se despliega en el orquestador y puede dar la información necesaria para la gestión del sistema, como se ve en la figura 8-7

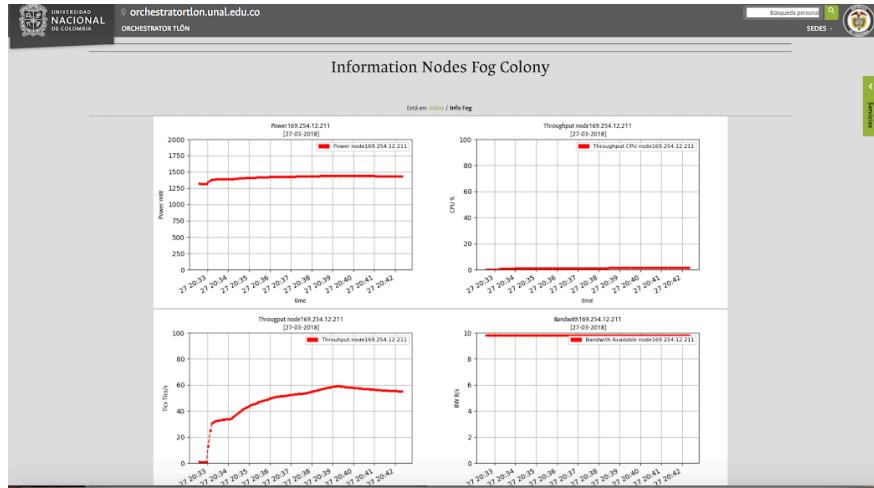


Figura 8-6: Consola Web SOVORA

```

  /--\ /--\ /\ /--\ /--\ /-\ \
  \ \ // \ \ \ / / / / \ \ / / \ \
  \ \ \ \_// \ \ / \ \_// \ \ \ \ \_\
  \_\_\_\_/\ \_\_/\_\_/\ \_\_/\_\_/\ \_\_
TLON Project 2018
Universidad Nacional de Colombia - Sede Bogota
[sovora#]

Documented commands (type help <topic>):
=====
EOF  clock  devices  exit  io  nodes  org  status
alive  cpu  disk  help  mem  nstatus  pid

[sovora#status
IDs nodes SOVORA : ['98b7401e-7f24-11e8-b633-b827ebfed7ae', 'e4780934-7f24-11e8-
a8b2-b827eb1f8ab6']
Bat Ip nodes SOVORA : ['169.254.4.218', '169.254.10.197']
Hosts on SOVORA : ['raspberrypi22', 'raspberrypi4']
Position nodes SOVORA : []
Sensors nodes SOVORA : []
Network nodes SOVORA : ["[TLONadhoc]"]
Boards nodes SOVORA : ['rpi_3']
sovora#

```

Figura 8-7: Consola de Comandos SOVORA

La información desplegada por el agente Local y por el Orquestador a manera de ejemplo se puede ver en las figuras 8-8 y 8-9 donde se evidencia el intercambio de información de estados y el monitoreo realizado, esta información es el insumo para las políticas del sistema junto con los logs de operación de cada uno de los miembros de SOVORA. Ejemplo de los logs del agente local y del orquestador se pueden ver en las figuras 8-10 y 8-11 respectivamente.

```
-----
      NODE STATE
-----
pow: 1317.86
% CPU: 0
thr_node: 0.0
BW used MB/s: 0.002288
BW available MB/s: 9.997712
-----
      Container 0
-----
thr_cont_0 : [0.0, 0.0]
mean_cont_0 : 0.0
thr_min 10.0
Cont-Controller 50000
BW used Mb/s: 0.002288
BW available Mb/s: 9.997712
-----
Delay _ Curr 0
-----
      Container 1
-----
thr_cont_1 : [0.0]
mean_cont_1 : [0.0]
thr_min 10.0
Cont-Controller 50001
BW used Mb/s: 0.002288
BW available Mb/s: 9.997712
-----
Delay _ Curr 0
-----
```

Figura 8-8: Agente Local SOVORA

```
-----
      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]
```

Figura 8-9: Orquestador SOVORA

El orquestador contiene una cola de tareas parametrizable para generar carga y tráfico sobre el sistema, de esta manera se validan diferentes escenarios posibles de operación y cargas sobre los nodos miembros de SOVORA.

```
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 /_ \ /__\ /\ /V__\/_\ /_\
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%)
```

Figura 8-10: Log Agente Local SOVORA

```
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 /_ \ /__\ /\ /V__\/_\ /_\
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%)
```

Figura 8-11: Log Orquestador SOVORA

En todos los escenarios se cuenta con esta información y los recursos respectivos para ser utilizados por el sistema. Bajo el modelo propuesto la institución desplegada es la presidencia del sistema

8.3.1. Escenario 1 - Monarquía

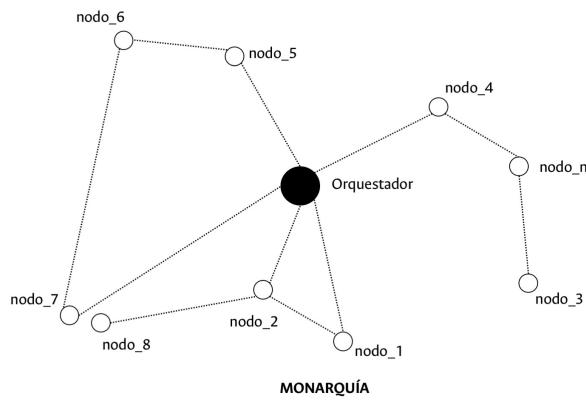


Figura 8-12: Modelo del escenario

En este modelo los agentes locales son desplegados en cada nodo y responden ante un solo nodo que tiene el rol de orquestador, cada agente local realiza el monitoreo continuo e intercambia información relevante para la toma de decisiones por parte del orquestador, como se evidencia en la figura 8-12. Este control permite mapear y gestionar los recursos del sistema, por ejemplo en la figura 8-13 se pude observar como se controla el limite del ancho de banda en el sistema, o en un nodo mediante la ejecución de comandos del agente local en cada nodo.

Cuando no se tiene gestión alguna del nodo se encuentran picos de trabajo en la operación de un nodo del sistema, este tipo de operaciones finalizan en consumos elevados de energía, saturación de CPU y bloqueos en el sistema operativo por la ejecución de aplicaciones con throughput que no corresponden a la capacidad de cómputo del dispositivo, a pesar de que la toma es sobre la interfaz inalámbrica se puede evidenciar un exceso en el envío de mensajes sobre el sistema que deriva en consumos elevados de energía y saturación de la red.

Al ejecutar políticas en el sistema estas permiten controlar el consumo de cpu y por ende la energía (figura 8-14), estas controlan la velocidad de la aplicación y el ciclo de reloj dentro de la cpu, del mismo modo afectan el consumo de memoria, el almacenamiento y ancho de banda dependiendo del tipo de aplicación.

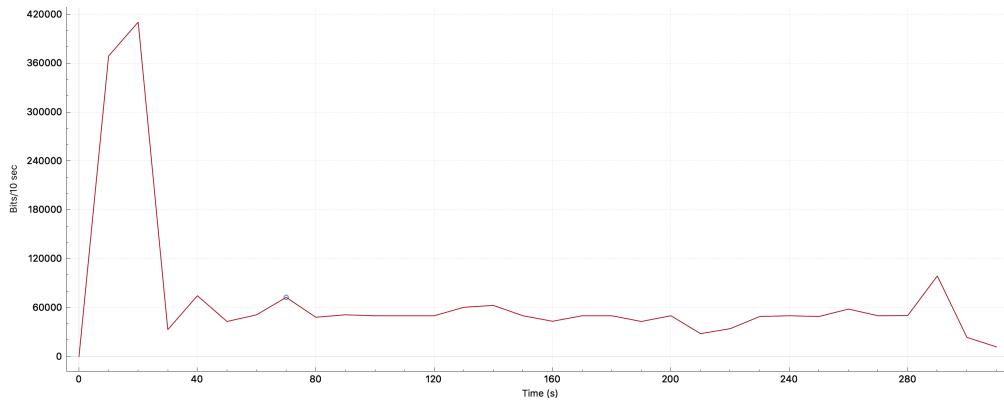


Figura 8-13: Control de Ancho de Banda

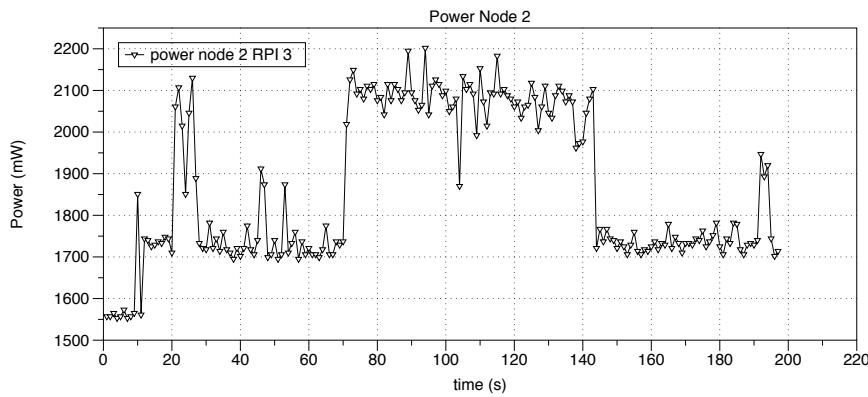


Figura 8-14: Energía Consumo de energía en un nodo

En la figura 8-16 se observa la gestión realizada por el sistema para mantener el throughput de la aplicación sobre un nodo, al controlar y estabilizar el consumo de recursos de cómputo, en este caso los ciclos de ejecución de una aplicación en un simple nodo, de esta forma se pueden programar tareas y generar aplicaciones distribuidas, el monitoreo se hace de forma local con un log en cada nodo y el log global del orquestador, la latencia es aproximadamente de un segundo después de tomar la medida del estado del nodo de forma local, sin embargo de ser requerido el log local puede ser enviado al orquestador para obtener el estado detallado de un nodo o un servicio particular.

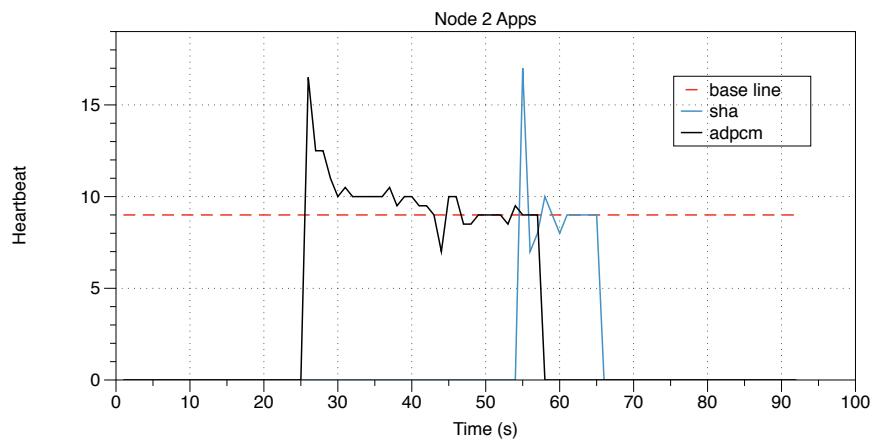


Figura 8-15: Throughput en de las aplicaciones en cada nodo

La gestión de los nodos dentro del sistema a partir de las políticas del orquestador y la caracterización de recursos utilizados por una aplicación, permiten controlar el consumo de recursos, los tiempos de operación y posibilitan determinar el tiempo de ejecución de una tarea dependiendo su entrada, del mismo modo conocer el consumo de energía, este esquema de operación presenta menos latencia y puede ser útil para aplicaciones que requieren tiempos de respuesta altos como lo son aplicaciones de video.

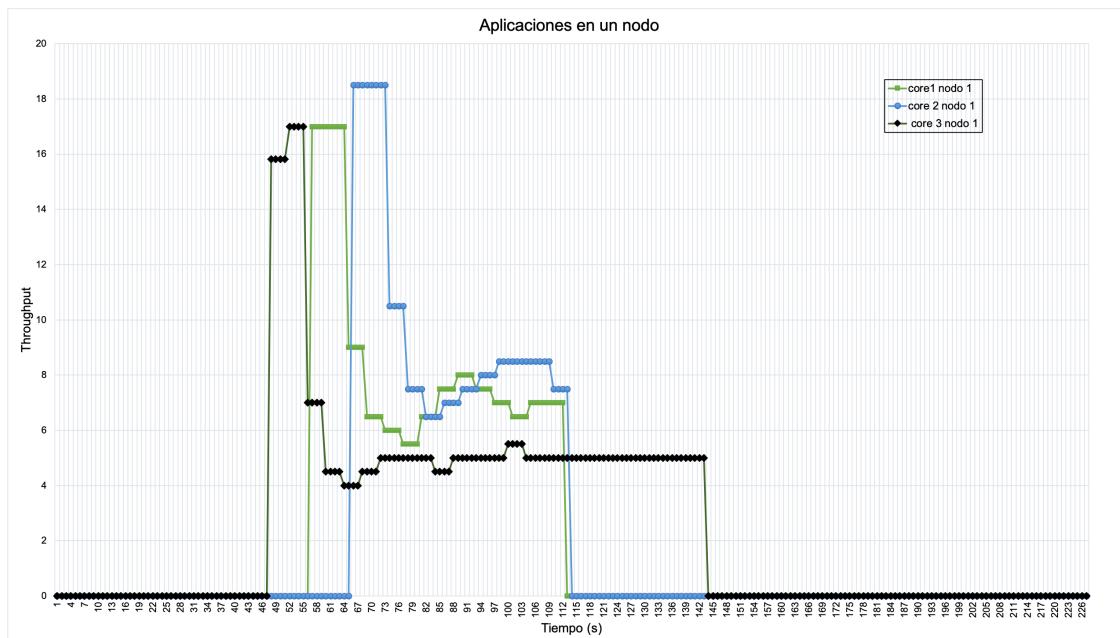


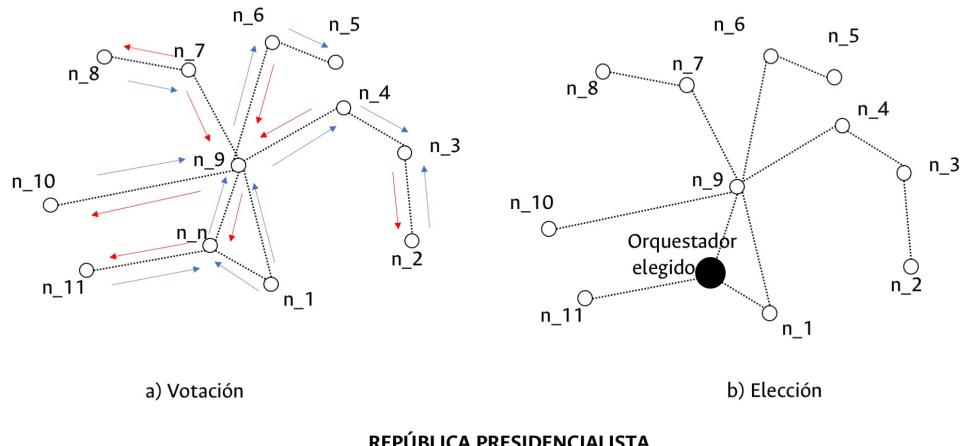
Figura 8-16: Throughput de las aplicaciones en un nodo

Medida	Core 1	Core 2	Core 3
Media	8,428571429	9,75	5,950874583
Desviación Estándar	3,627456624	4,071175271	0,295306564
Varianza	6,020833333	16,57446809	0,087205967

Tabla 8-22: Análisis Estadístico Figura 8-16

Como se evidencia en la tabla 8-22, en el core 3 del primer nodo se cuenta con una desviación estándar menor, acorde con la media y el valor esperado de la línea base, la cual es cinco, es importante resaltar que la línea base representa el throughput deseado durante la ejecución del experimento dadas la decisión tomada por el orquestador y el tiempo de ejecución, en este caso el tiempo de ejecución sobre este core es más alto por lo cual el valor es más cercano a la media.

8.3.2. Escenario 2 - República Presidencialista

**Figura 8-17:** Modelo del escenario

A diferencia del escenario presentado en la sección 8.3.1, existe un cambio de rol constante del orquestador en el sistema esto permite cambiar el modo de operación de un dispositivo dentro de una aplicación específica teniendo en cuenta por ejemplo, la cantidad de recursos requerida, el balanceo de recursos dentro de los miembros o el balanceo de tareas básicas para la operación del sistema virtualizado. Es importante resaltar que los logs del sistema también son migrados al igual que el rol

del nodo, este procedimiento usa el agente local descrito en la sección 7.5.7.3, este elemento permite transportar la información entre los diferentes nodos del sistema como un servicio nativo para realizar la distribución, cambio de roles y demás tareas necesarias para mantener el sistema.

Este escenario permite realizar una elección del nodo orquestador entre los miembros del sistema, al igual que distribuir la carga de los nodos y su rol dentro del sistema, bajo la arquitectura propuesta es posible tener un nodo con los roles de orquestador y de agente local de forma simultanea, al igual que tener varios agentes locales en ejecución.

En las figuras **8-18** y **8-19** se muestra la ejecución distribuida de tareas en los nodos miembros del sistema , estas tareas reservan recursos de cómputo para la ejecución de las aplicaciones, enviar resultados al nodo que lo requiere y guardar la información generada en el espacio de disco reservado para S.O.V.O.R.A., en este escenario se desplegaron 6 tareas distribuidas en los nodos miembros cuatro (4) Raspberry pi modelo 3, un (1) raspberry zero y un (1) odroid, este escenario busca distribuir la carga de aplicaciones por el nodo que haga las veces de orquestador y controlar la ejecución de las aplicaciones con un throughput específico limitando el consumo de energía. Esta distribución permite reducir el tiempo de ejecución de una tarea y controlar el consumo de recursos, uno de los problemas detectados en la ejecución de este modelo es la latencia y pérdida de control momentánea al cambiar de orquestador el modelo permite distribuir carga pero no es confiable para aplicaciones que requieran de datos en tiempo real y sincronización de servicios, debido a la latencia generada y la sincronización en el cambio de rol del nodo dentro del sistema. En el recuadro se ve la ejecución en el mismo instante de tiempo de aplicaciones con diferente throughput, en particular el evento en el cual una aplicación puede fallar en la ejecución o el nodo es bloqueado, es importante resaltar que al no existir control sobre el consumo de recursos dentro del sistema por lo general busca consumir el 100 % del recurso que requiera ya sea cpu, memoria e incluso disco, este modo de operación puede ser útil para servicios qu no requieran publicación y transaccionalidad alta en la entrega de data al usuario o aplicación, por ejemplo la publicación de un portal web.

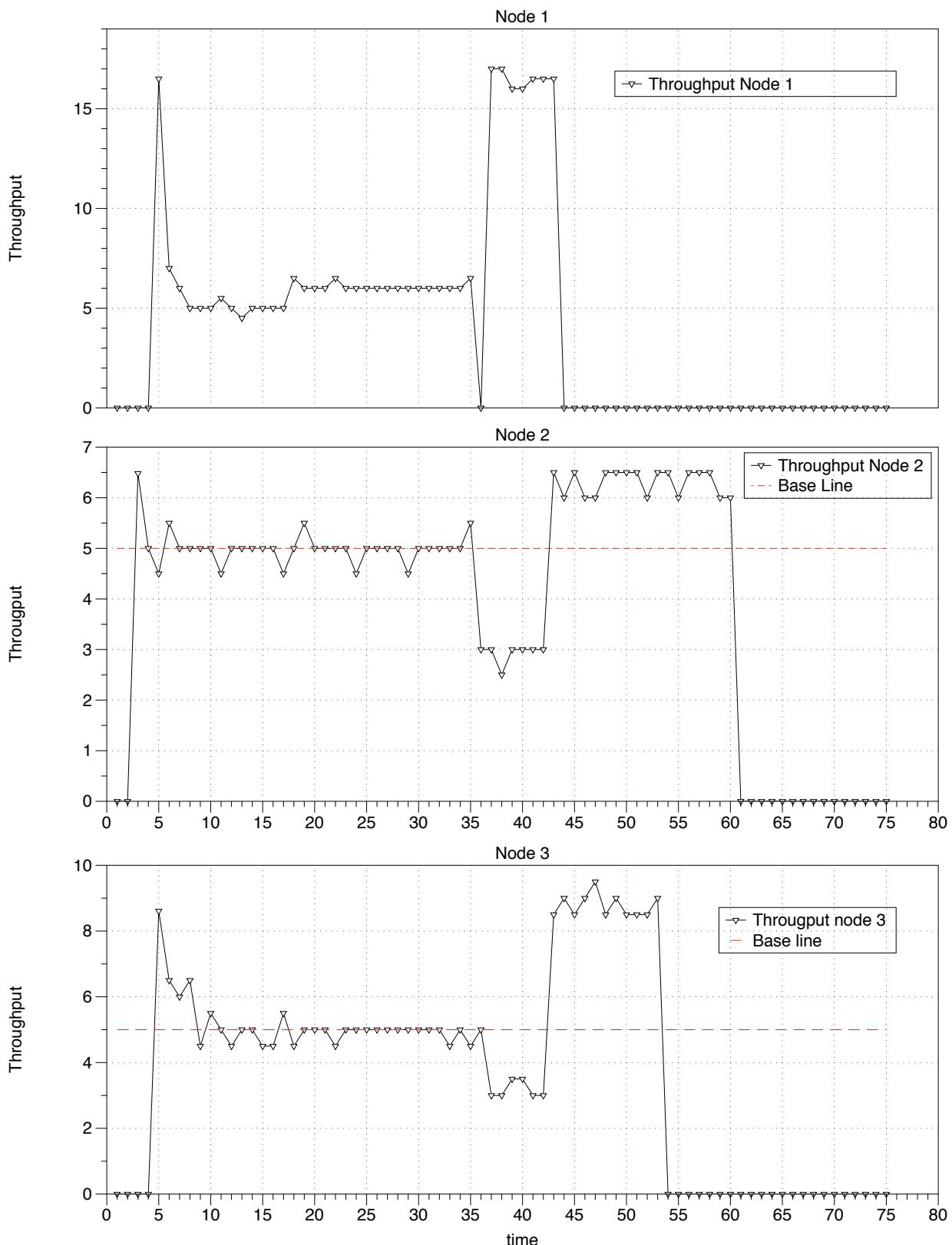
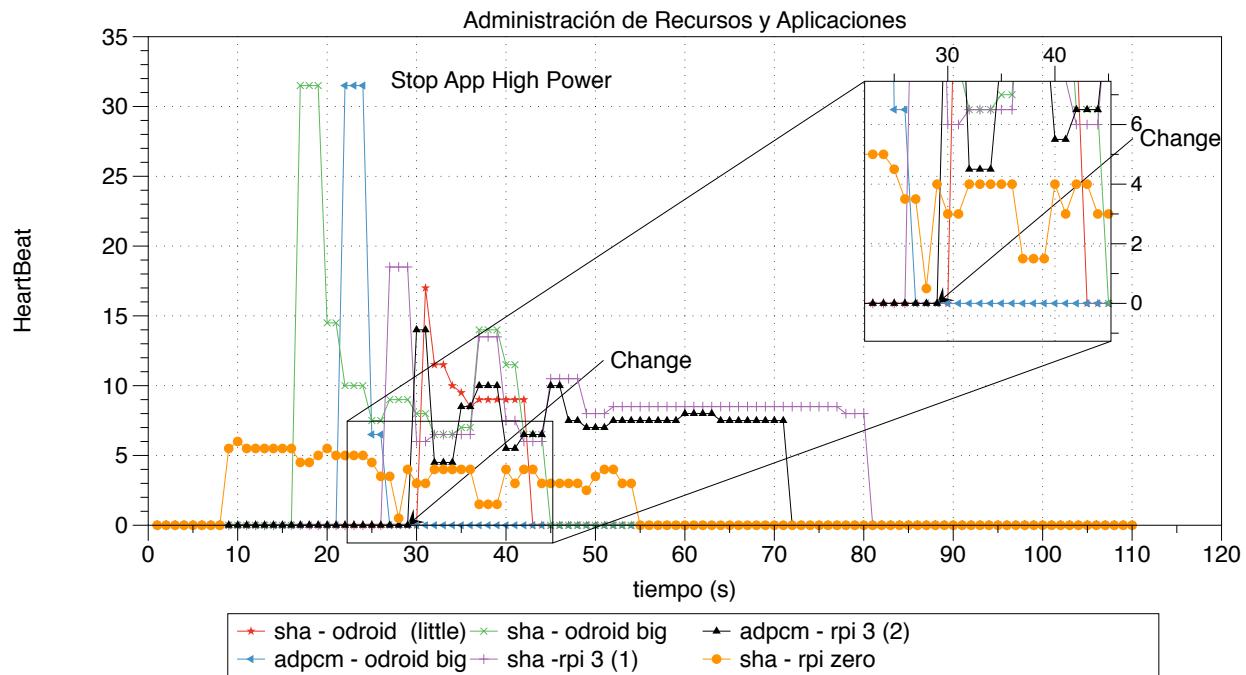
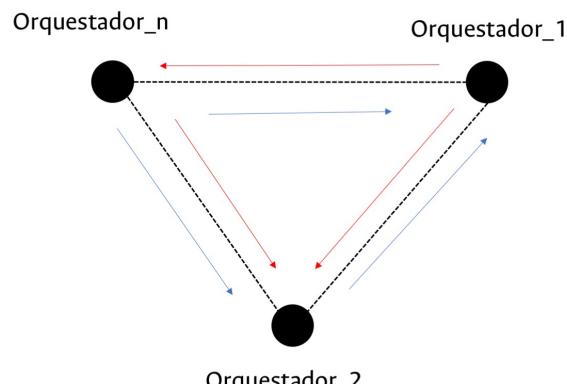


Figura 8-18: Control de los nodos

**Figura 8-19:** Modelo del escenario

8.3.3. Escenario 3 - República Parlamentaria

**Figura 8-20:** Resultados de operación del escenario

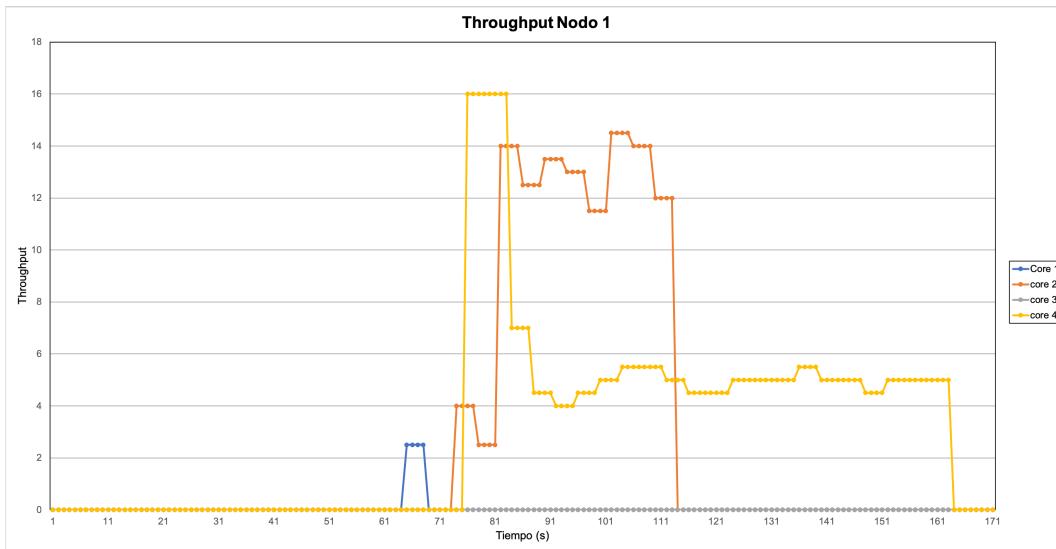


Figura 8-21: Resultados de operación del escenario nodo 1

Medida	Core 1	Core 2	Core 3	Core 4
Media	2,5	11,15	0	6
Desviación Estándar	0	4,114078393	0	3,230600812
Varianza	0	16,92564103	0	10,43678161

Tabla 8-23: Análisis Resultados nodo 1

En el nodo 1 se logró tener control del core 1 en su totalidad, con la aplicación de decodificación de imágenes jpeg, y un control parcial del core 2 en cuanto a gestión de recursos, para el core 3 según la política desplegada no se uso para este escenario, en cuanto al core 4 se obtiene un mejor desempeño debido al tiempo de ejecución de la aplicación sobre este nodo.

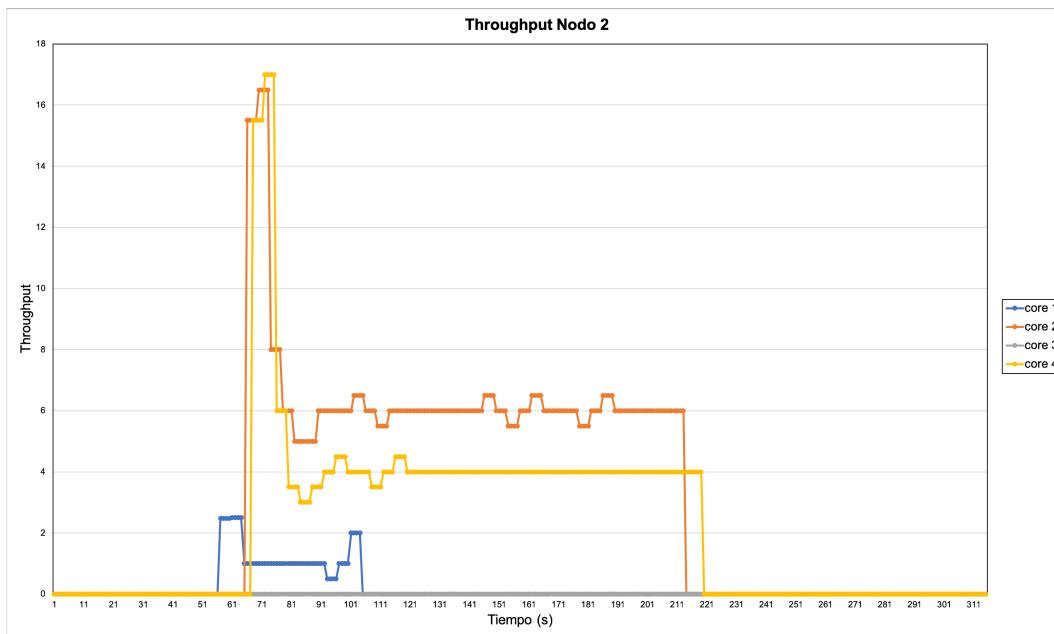


Figura 8-22: Resultados de operación del escenario nodo 2

Medida	Core 1	Core 2	Core 3	Core 4
Media	1,290129667	6,554054054	0	4,657894737
Desviación Estándar	0,63143912	2,314649542	0	2,776689303
Varianza	0,398715362	5,3576025	0	7,710003486

Tabla 8-24: Análisis Resultados nodo 2

En este nodo se obtuvieron mejores resultados en todos los cores al distribuir la tarea compartida, es importante resaltar que los cuatro nodos hacen parte de una sola aplicación desplegada en el sistema SOVORA, acorde a la política desplegada por el orquestador en el sistema, como se ve en las gráficas de esta sección el throughput converge al nivel deseado y retorna los recursos una vez la tarea o la ejecución del algoritmo termina.

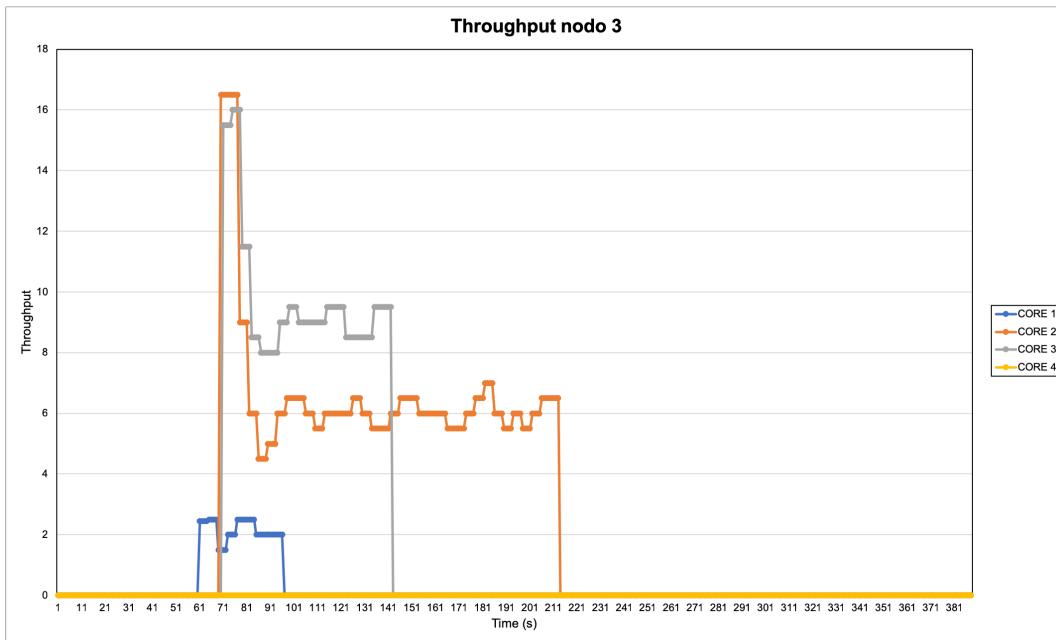


Figura 8-23: Resultados de operación del escenario nodo 3

Medida	Core 1	Core 2	Core 3	Core 4
Media	2,160637222	6,638888889	9,805555556	0
Desviación Estándar	0,332340761	2,496462143	2,252628442	0
Varianza	0,110450381	6,232323232	5,074334898	0

Tabla 8-25: Análisis Resultados nodo 3

En este nodo se obtienen resultados favorables para el despliegue de las aplicaciones, sin embargo es importante resaltar la similitud con el desempeño del nodo 2, esto se debe a la distribución de cargas bajo nodos con similares características físicas, o en cuanto a recursos de cómputo. Esta información es parte de las tablas en tiempo de diseño del orquestador.

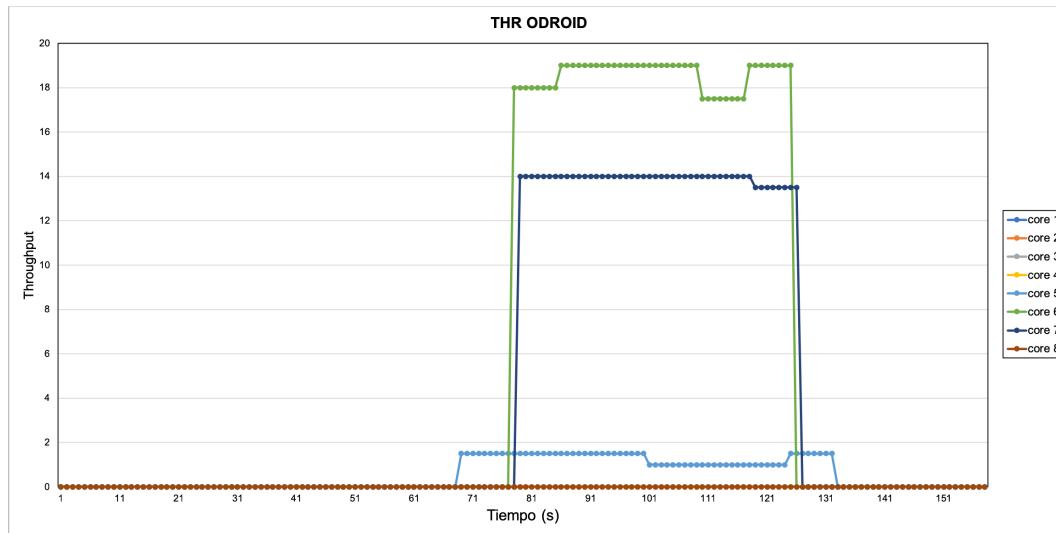


Figura 8-24: Resultados de operación del escenario nodo 4

Medida	core 5	core 6	core 7
media	1,3125	18,58333333	13,91666667
desviación estándar	0,243975018	0,613095853	0,188310894
varianza	0,05952381	0,375886525	0,035460993

Tabla 8-26: Análisis Resultados nodo 4

En el nodo 4 de este escenario tiene un comportamiento diferente dadas las condiciones del dispositivos odroid y el uso de una política que permite variar la frecuencia de los cores de este dispositivo según su familia, en este caso los recursos manipulados están de lado de la familia Big del procesador del odroid, dando más estabilidad a las aplicaciones a pesar del tiempo de ejecución.

Este escenario permite a todos los nodos tener el rol de orquestador, es decir se tiene un consenso del resultado al momento de distribuir una tarea sobre los nodos del sistema, este escenario usa los puertos de comandos y de consenso para realizar las tareas sobre el sistema, una de las ventajas de este enfoque es el modelo compartido de logs y el registro de todas las tareas sobre el sistema, generando un nivel de redundancia en el sistema.

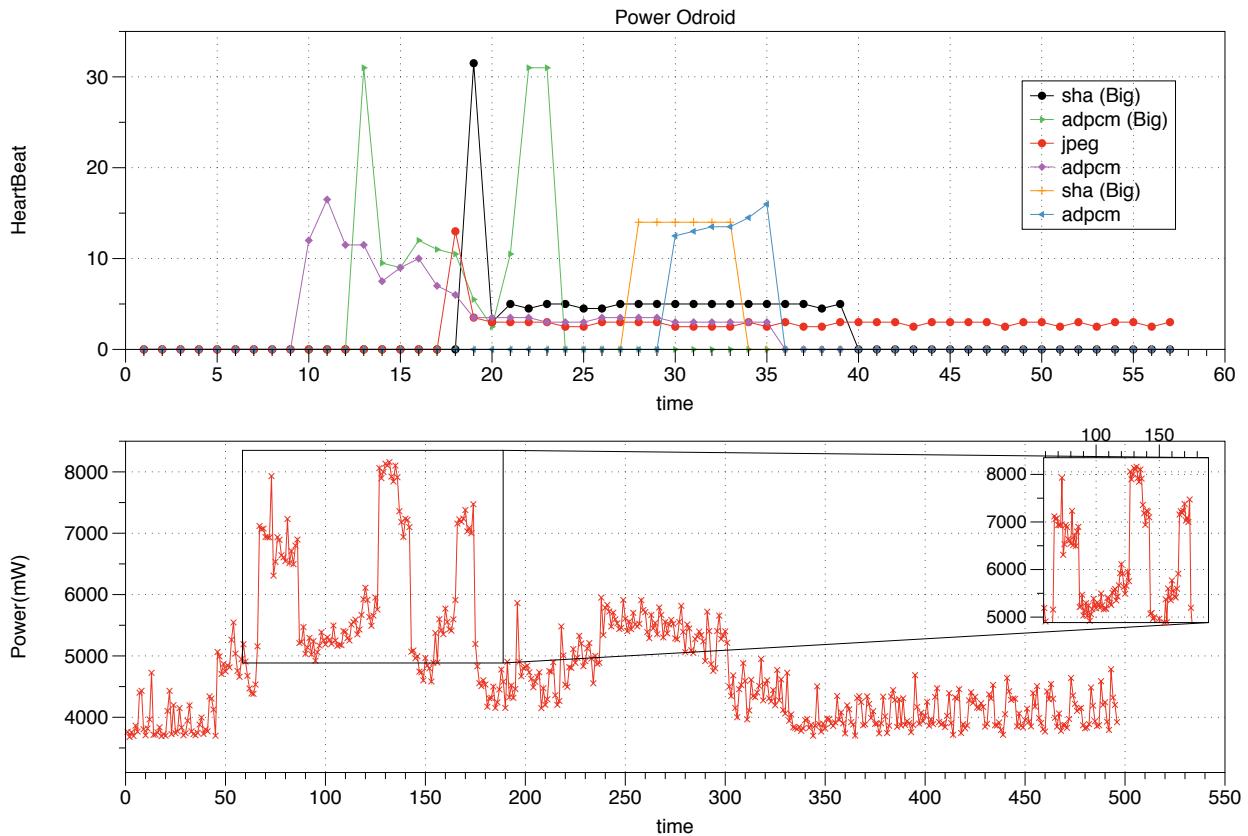


Figura 8-25: Control de los nodos

El manejo de la aplicaciones se realiza de forma análoga al escenario mostrado en la sección 8.3.1, el cambio de modelo influye en el retardo de las transmisiones y los flujos de tráfico generados por las aplicaciones ejecutadas sobre este modelo una de las mediciones claves sobre este modelo es la latencia entre los nodos y la entrega de información al nodo destino. En el modelo se realizó el balanceo de tareas en un nodo de acuerdo a la elección de los orquestadores en este caso en el nodo con más recursos el Odroid, este escenario es mostrado en la figura 8-25 y su respectivo consumo de energía, del mismo modo en un dispositivo Raspberry Pi como se ve en la figura 8-26 este esquema de operación distribuye aplicaciones en nodos con más recursos y con la disponibilidad de los mismos para la ejecución de aplicaciones, en este caso el nodo 2 ejecuta las tareas con más necesidad de recursos del sistema, en la anterior gráfica (8-25), toda la carga del sistema es soportada por este nodo le cual posee la capacidad y el consumo de energía no es alto en comparación con el modelo compartido, sin embargo los servicios que requieren respuesta rápida pueden verse penalizados por los tiempos de consenso, aunque el algoritmo usado para el consenso es sencillo, genera latencia en la distribución de tareas en el sistema.

Cada uno de los experimentos presentados en este trabajo fueron realizados 20 veces por escenario

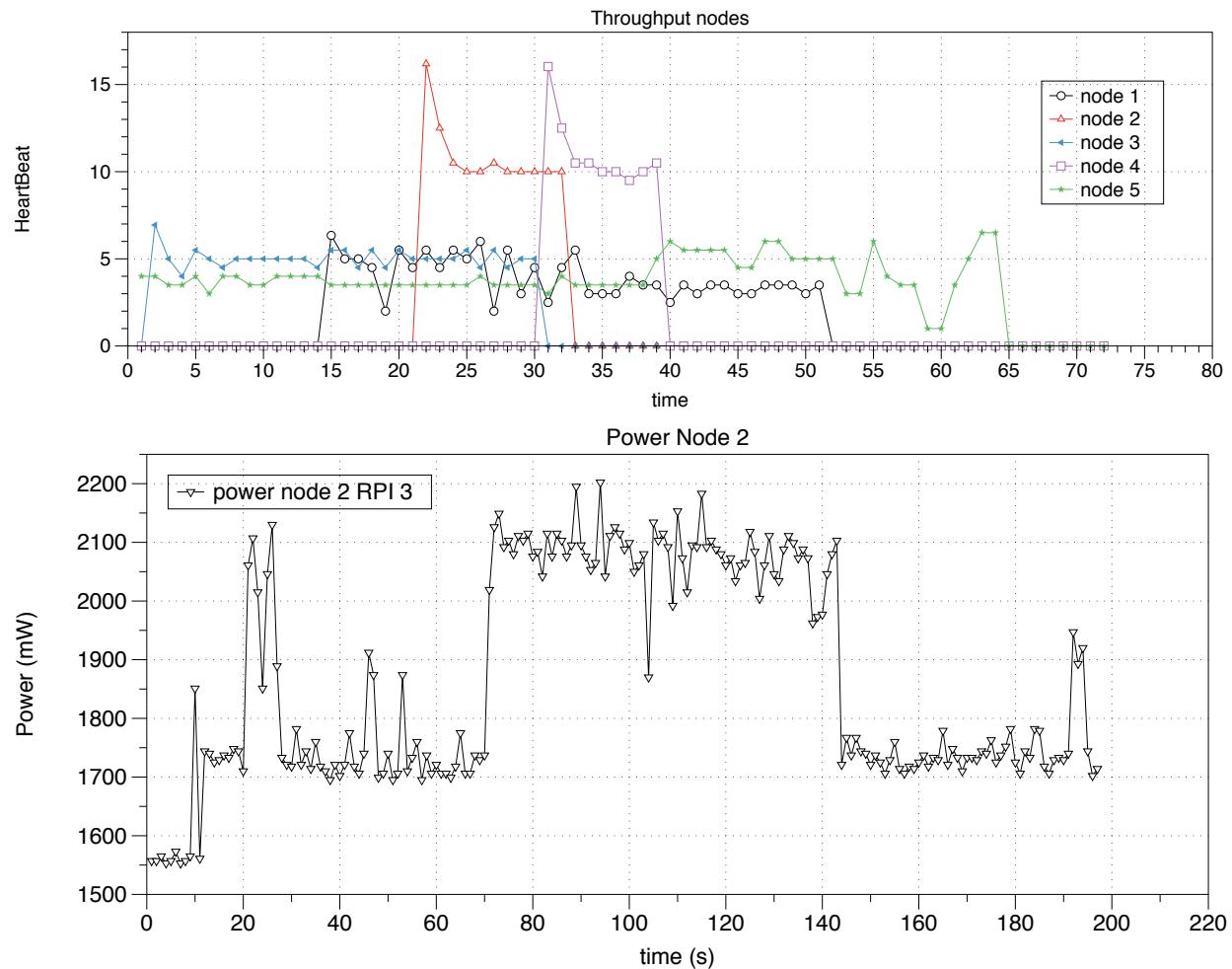


Figura 8-26: Control de los nodos

gestionado por el orquestador, para una ejecución que involucra 3 nodos con Throughput de 8, 11 y 7:

Medida	Nodo1	Nodo2	Nodo3
Media	8,881396045	10,55847272	7,098284959
Desviación Estándar	2,005431584	2,091430928	2,415926747
Varianza	4,021755836	4,374083329	5,836702047

Tabla 8-27: Análisis Estadístico Orquestador

En este apartado la desviación estándar esta dada por el control de los picos iniciales de cada uno de los consumos de recursos de los nodos, al ejecutar alguna aplicación, como media de tiempo de estabilización tenemos un tiempo promedio de 2,3 segundos en cuanto a procesamiento, para los recursos de memoria y almacenamiento los recursos son constantes dada las características de las aplicaciones y del experimento sin embargo son monitoreados durante la experimentación.

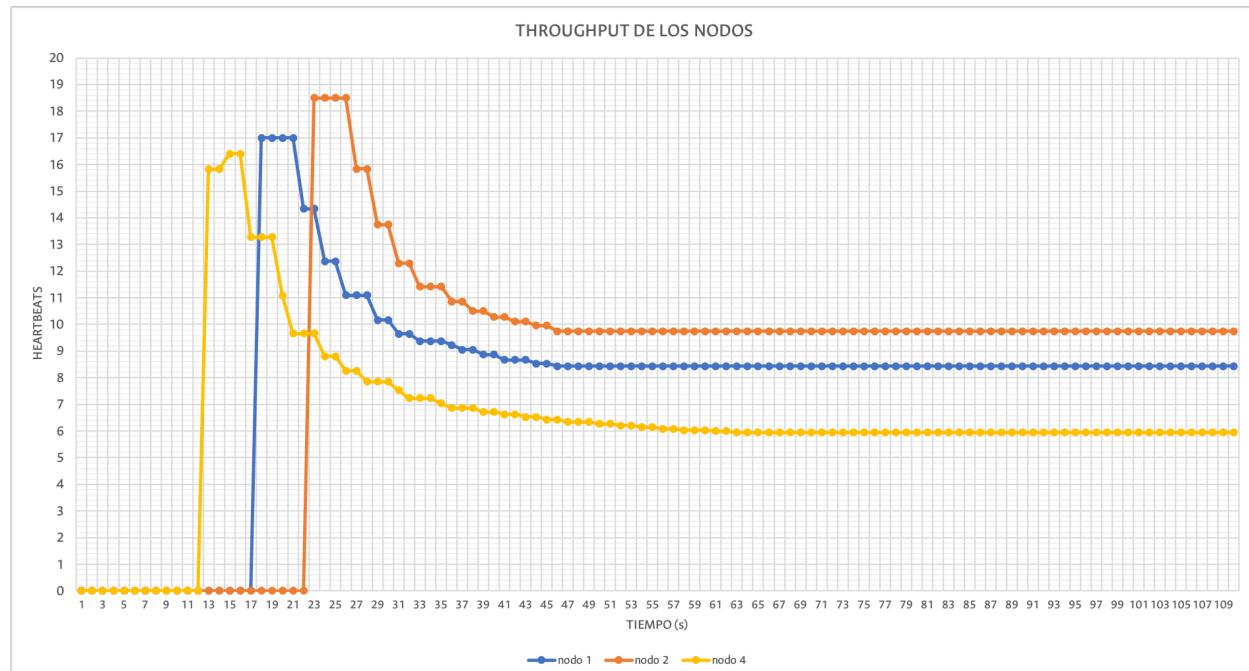


Figura 8-27: Media de recurso CPU vista desde el orquestador

El consumo de recursos converge al throughput deseado, esta información es monitoreada por el orquestador según el tiempo parametrizado en el sistema para el efecto de este experimento es de 3 segundos. Como se ve en el gráfico, converge al valor deseado o estipulado en la política, este valor es entregado por el agente local quien periódicamente calcula la media de sus consumos y alimenta

un log con esta información para el histórico de rendimiento, insumo clave para el orquestador en su toma de decisión.

Capítulo 9

Conclusiones y Recomendaciones

9.1. Conclusiones

El desarrollo del sistema TLÖN continua de la mano de estos primeros resultados y artefactos computacionales presentados en este trabajo, ahora continua con la integración de los demás componentes desarrollados en otras tesis de doctorado, maestría y pregrado. Este diseño modular permite escalar la solución y manipular de diversas formas las abstracciones propuestas, creando el Sistema Distribuido TLÖN.

Las conclusiones más relevantes de este trabajo son las siguientes:

- a)** El mapeo de recursos (Cómputo, Red, Almacenamiento y dispositivos de I/O) sobre una red inalámbrica permite generar la capa de abstracción virtualizada, la distribución de tareas sobre el sistema , permite la inclusión de políticas que no involucren un alto rendimiento, exclusión de usuarios o sobre carga de tareas en el sistema de cómputo, el prototipo desarrollado permite proporcionar al usuario una calidad de servicio a pesar de las limitaciones de los dispositivos de los usuarios. Este mapeo realizado localmente por el agente local y consolidado en los nodos orquestadores según el modelo de pseudo estado, permiten acceder a estos recursos distribuidos virtuales, y realziar estas tareas a pesar de la latencia de las comunicaciones.
- b)** Los escenarios de prueba propuestos (Monarquía, República Presidencialista, República Parlamentaria), son tres modelos de pseudo estado, en este sistema artificial controlable han sido evaluados y se ha demostrado su viabilidad como mecanismo de gestión de recursos , asignación de tareas y despliegue de aplicaciones sobre una red Ad hoc, pero definiendo sus limitaciones para desplegar servicios específicos , como por ejemplo el modelo de República parlamentaria al tener mayor tiempo de despliegue de las tareas, no es adecuado para un servicio de

respuesta en línea como vídeo por demanda.

- c) El modelo pseudosocial basado en la institucionalidad involucra tipos de Estado y formas de gobierno en sistemas artificiales, esta inclusión genera una estructura donde pueden existir agentes bajo reglas adoptadas análogas a las reglas de los hombres, a pesar de que este modelo no es implementado en su totalidad en este proyecto, el hecho de definir las interacciones entre los recursos físicos, las interacciones de los agentes y las abstracciones de las instituciones básicas como la presidencia o monarquía permiten identificar los elementos básicos para el desarrollo de esta modelo de gestión de aplicaciones y recursos basados en la institucionalidad.
- d) Los sistema ubicuos como las redes ad hoc, sistemas distribuidos como el Sistema TLÖN pueden ser modelados con álgebras de procesos o como en este caso con Bigrafos, modelo de especificación de interacciones en espacios físicos y lógicos, modeladas como reglas de interacción, esta forma de representación permite identificar las interacciones a nivel de Sistema, de nodo, de agente e incluso del código en ejecución, la versatilidad de los bigrafos permite modelar interacciones aún más complejas que no se pueden detectar con pocos nodos, e incluso servir como modelo abstracto base para simular sistemas altamente distribuidos.
- e) El sistema de cómputo S.O.V.O.R.A. se ha desarrollado para operar a nivel de usuario, servidor y nube, este modelo permite la escalabilidad y la elasticidad tanto en recursos como en la distribución de aplicaciones en los ámbitos definidos por el sistema y el usuario, este modelo interactúa con arquitecturas como lo son IoT, Fog Computing o Micro Data Center. Acorde con las necesidades de los sistemas, las aplicaciones y los usuarios de estos ecosistemas de cómputo.
- f) La operación de sistemas dinámicos inalámbricos permite eliminar carga a los sistemas del núcleo de la red o sistemas donde se alojan las aplicaciones de forma nativa, debido a que la mayoría de estos dispositivos se encuentran en el borde la red y requieren de aplicaciones o servicios de otras redes, es necesario tener un elemento de control que pueda operar ya sea de forma centralizada o distribuida que deleguen ciertas tareas de su operación a la abstracción denominada orquestador.
- g) El manejo de clústers físicos y lógicos permite identificar nuevas interacciones entre los nodos y dispositivos, para los procesos de asignación de tareas, monitoreo de recursos y gestión del sistema, de una forma más granulada y manteniendo subprocessos identificados para la detección de fallas, incluso generando esquemas de cooperación internos por nodo.
- h) Las aplicaciones para este tipo de sistemas deben ser desarrolladas con el esquema de micro servicios para realizar una partición y distribución de las aplicaciones en los diferentes miembros del sistema, es decir se debe crear un modelo de aplicación distribuida dinámica para la ejecución de los microservicios de cada aplicación en el sistema.

- i) Es necesario realizar el diseño de imágenes a la medida y con capacidad de dispersión y ejecución por capas, para el despliegue de los servicios y/o aplicaciones ya que estos procesos solo se pueden dar con el contenido completo de la imagen que en algunos casos consumo demasiados recursos para ser ejecutada por un nodo .

9.2. Recomendaciones y Trabajo futuro

A lo largo de esta investigación se han resuelto dudas, corroborado supuestos y en el mismo sentido desecharo una gran cantidad de modelos e ideas que no son posibles de realizar con los artefactos computacionales actuales o simplemente no eran tan eficientes y necesarios como se pensó en la fase de diseño. Las siguientes son las recomendaciones y trabajo futuro derivadas del desarrollo de S.O.V.O.R.A., abordadas en su mayoría desde temáticas fuera del alcance de esta investigación:

- a) Abordar con más profundidad el tema de seguridad de la información ya que la solución realizada contiene pocos parámetros de seguridad y depende bastante del sistema operativo donde se despliega.
- b) Agregar un componente de Network Coding en las transmisiones para mejorar el throughput y las condiciones generales del sistema en su despliegue y operación.
- c) Crear un modelo de simulación de S.O.V.O.R.A. que permita validar el sistema desarrollado con mas dispositivos , tráfico y aplicaciones desplegadas, un ambiente de simulación como Ns-3 es viable, aunque se crearon fuertes abstracciones no es fácil prever fallas que se encontraron durante la implementación del sistema, del mismo modo hacer test con más nodos o con diversos escenarios de pruebas.
- d) Ahondar en la interoperabilidad de la solución con otros orquestadores vigentes y altamente comerciales, al igual que con otros sistemas multi agente y aplicaciones del mercado.
- e) Proponer aplicaciones más robustas y necesarias en la industria en ecosistemas como Internet de las cosas, ciudades inteligentes, Smart Farming y otras tecnologías derivadas para obtener resultados del comportamiento de estas aplicaciones en este sistema.
- f) Realizar un modelo de operación del sistema como módulo del kernel de Linux y reducir la carga computacional asignada al lenguaje de alto nivel
- g) Crear modelos de instituciones sobre el sistema como seguridad, control de recursos físicos y lógicos que tomen los datos generados por la capa de virtualización y desplieguen servicios especializados tanto de control como de aplicación en el diferentes niveles computacionales.
- h) Refinar la interconexión con otras redes de la solución propuesta y escalarla entre diversas arquitecturas de cómputo, es decir a nivel de servidores, data center y cómputación en la nube.
- i) Validar la operación de este modelo con otros protocolos de comunicación como Bluetooth, Zigbee, Ethernet y verificar el impacto de las políticas , anchos de banda e interacciones del sistema con trazas diferentes a las usadas por el estándar IEEE 802.11.

Apéndice A

Redes Ad hoc

La proliferación de las comunicaciones y las redes en general deben su desarrollo exponencial a la aparición del componente inalámbrico en dispositivos de cómputo, los cuales han permitido la difusión de servicios y nuevas formas de interactuar entre las tecnologías y los usuarios. Shahzadi y cols. (2017)

La evolución de las interfaces inalámbricas han permitido pasar de un sistema monoservicio a esquemas múltiples compartidos, donde inicialmente se tenía un dispositivo con una sola interfaz inalámbrica, hoy tenemos dispositivos móviles con varias interfaces: una interfaz 4G, una interfaz Wifi y una interfaz bluetooth en la mayoría de los casos las cuales son canales potenciales de comunicación para prestar servicios, bajo estas condiciones se pueden generar redes superpuestas es decir tener ámbitos diferentes de operación para cada interfaz con el fin de prestar o recibir algún tipo de servicio solicitado por el usuario.

Otro aspecto importante es la aparición de las redes sociales, las cuales modificaron el comportamiento de los usuarios y cambiaron sus necesidades de conectividad, junto con sus comportamientos sociales y las interacciones con los dispositivos móviles. En este sentido han cambiado las necesidades de conectividad, requiriendo novedosas formas para el despliegue de los servicios solicitados en sus dispositivos.

Bajo este esquema inalámbrico, móvil y dinámico aparecen las redes Ad hoc. ¿Qué es una red Ad hoc? es una red de computadores conectada por interfaces inalámbricas, con un nivel de recursos dinámico, capaces de proveer servicios sin importar las condiciones dinámicas y estocásticas de los nodos al transcurrir el tiempo. Dos propiedades deseables en este tipo de sistemas son: la primera de ellas la auto-organización, y la segunda, no menos importante que la anterior la de tener una infraestructura

descentralizada. Por lo anterior este tipo de sistemas pueden ser capaces de generar comportamientos pseudosociales desde el instante de su conformación hasta el fin de su operación. Formalmente las redes Ad Hoc son un grafo aleatorio Loo y cols. (2016); Ilyas (2002); Barbeau y Kranakis (2007); Basagni y cols. (2004) con un conjunto de vértices, comúnmente llamados nodos, en este caso móviles, unidos por un conjunto de enlaces denominados aristas, que cambian de forma dinámica en función del tiempo y las condiciones del ambiente; por ejemplo las peticiones de los usuarios.

$$\text{Definición formal de Manet} \quad M = N, L, G_p(l), I \quad (\text{A-1})$$

Dando más profundidad a la definición expuesta una red Ad hoc móvil (MANET en inglés) se puede definir como un conjunto de nodos (N), enlazados por un grupo de enlaces L , con un conjunto de interacciones I , todos ellos incluidos como un multi-grafo aleatorio $G_p(l)$, como se ve en la ecuación A-1.

Es posible incluir la operación de redes Ad hoc dentro de un ecosistema dinámico como lo es IoT, Fog Computing e incluso las redes de sensores, en este contexto se pueden identificar tres estados de operación dentro de una red de este tipo **la formación** donde se crea la red, **la operación** donde se establecen las conexiones de la red y se tienen las tablas de enrutamiento actualizadas y los servicios activos, finalmente **el mantenimiento** donde se tienen servicios desplegados a pesar de las condiciones cambiantes de la red, entrada y salida de nodos, pérdidas de enlaces entre otros, esta etapas se pueden ver en la figura A-1

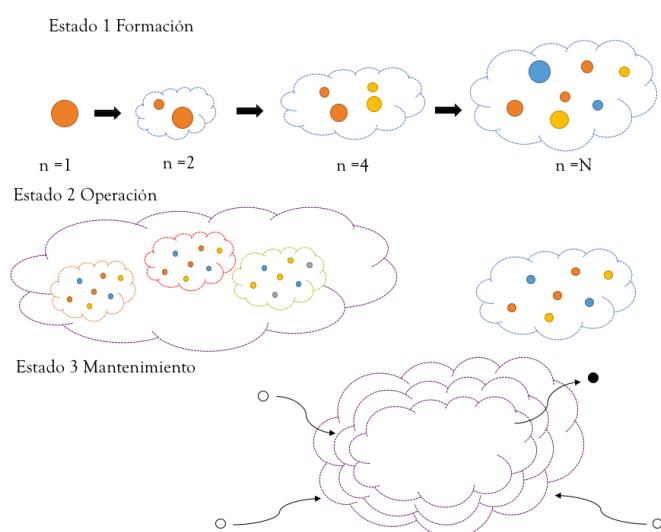
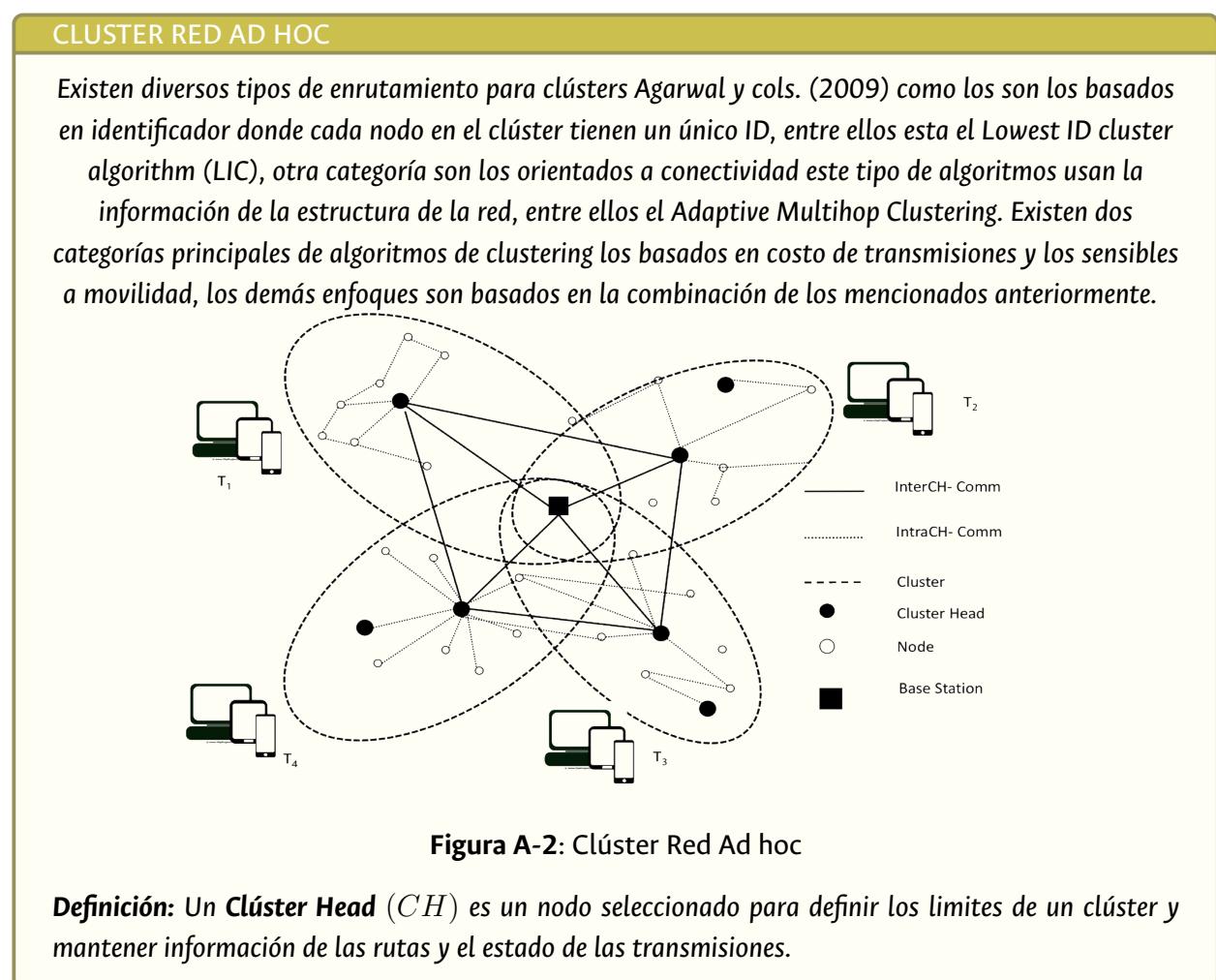


Figura A-1: Red Ad hoc

A.1. Clústers en Redes Ad hoc

Un escenario de operación para MANET son los Clústers Touati y cols. (2017), los cuales se basan en una organización jerárquica, en un conjunto de dispositivos. Cada clúster es un conjunto de diferentes nodos, de los cuales uno es el coordinador o representante conocido como Clúster Head (*CH*) y el otro el nodo gateway. El nodo coordinador permite a los nodos miembros comunicarse con otro clúster u otras redes, el *CH* es el responsable de administrar la comunicación intra e inter clúster Touati y cols. (2017); Xiao y Pan (2009); Ramanathan y cols. (2010); Yu y Chong (2005).



Las comunicaciones internas del clúster permiten a todos los nodos miembros intercambiar servicios y mensajes sobre ellos, compartir estados de difusión y transmisiones de datos efectivas. Por otro lado, la comunicación entre clústeres solo son intercambios entre *CH*, en algunos casos los *CH* se utilizan para habilitar comunicaciones de largo alcance como sistemas celulares o WiMAX, Mesh, entre

otros o para mejorar la cobertura, estas comunicaciones crean la posibilidad de pasar mensajes a otros vecindarios.

La comunicación entre clústers Vinayakray-Jani y Sanyal (2012) está asegurada por los nodos de borde quienes trabajan como gateway, por razones de seguridad, no todos los nodos fronterizos pueden garantizar el enlace entre dos grupos, pero necesitan tener un alto nivel de confianza para ser los nodos de intercambio entre clúster adyacentes. Otro elemento clave es el protocolo de enrutamiento, el cual distribuye la carga, el flujo de mensajes e incluso puede ser señalizado en la capa MAC para la transmisión de paquetes intra e inter clúster Gavalas y cols. (2006).

A.2. Nubes Móviles

La interacción entre las interfaces inalámbricas, la difusión de Internet y las necesidades de los usuarios han generado nuevos modelos de red, donde las redes superpuestas están presentes en todo los ámbitos de usuario, desde una plataforma centralizada monolítica hasta un sistema altamente dinámico y estocástico. Estas interacciones involucran los medios de transmisión, la infraestructura y el usuario, los servicios están en la intersección de estos tres elementos, esto gracias a las necesidades de conectividad y movilidad de los usuarios.

Podemos mostrar una primera aproximación a las nubes móviles como un arreglo cooperativo de nodos conectados compartiendo oportunisticamente recursos, esto se puede ver como un sistema distribuido clásico. Una segunda definición incluye elementos de infraestructura necesaria para su despliegue: una nube móvil es un arreglo cooperativo de dispositivos inalámbricos cercanos, cada uno puede ser conectado con otras redes mediante puntos de acceso o estaciones bases.

Nube Móvil

Definición: Una nube móvil es una plataforma computacional flexible, dinámica y estocástica que gestiona recursos de cómputo distribuidos que son enlazados inalámbricamente; estos a su vez, pueden ser cambiados, movidos, aumentados y en general, combinados de formas novedosas Fitzek y Katz (2013)

A.2.1. Nubes Sociales

En un contexto más amplio involucrando las necesidades y preferencias del usuario una nube móvil o una nube de cosas Nan y cols. (2018) es una plataforma flexible para establecer redes sociales móviles, es decir, redes donde interactúan usuarios que tienen la libertad de ser móviles.

En este sentido surge una interacción directa con las preferencias sociales de los usuarios y de las necesidades de los mismos, no obstante surge un interrogante sobre ¿Qué es una sociedad o como podemos definirla?. Una sociedad es una asociación, más o menos autosuficiente, de personas que en sus relaciones generan comportamientos de colaboración y cooperación con el fin de obtener bienestar y/o felicidad. Sus miembros reconocen ciertas reglas de conducta como obligatorias y en su mayoría están de acuerdo con ellas. (Rawls, 2012).

Las nubes móviles como se ha indicado son cooperativas, esta es la capa base donde se despliegan este hecho hace que sus miembros acepten reglas mínimas para ingresar a este sistema. se pueden definir dos dominios de cooperación dentro de las nubes móviles el primero el dominio técnico donde podemos tener cooperación forzada y permitida por la tecnología (Hardware),y el segundo dominio denominado social se tienen el altruismo y la cooperación permitida socialmente.

Estos esquemas se basan en la relación costo-beneficio, quien valora esta relación es el usuario o propietario del dispositivo, quien instala, modifica, opera y distribuye servicios desde su dispositivo móvil.

La relación es simple, que costo (C) se paga y cual es el beneficio (B) obtenido, estas son las descripciones de cada una de ellas:

1. Un ejemplo de *cooperacion forzada* son las redes de sensores donde la relación costo beneficio es $C > B$ o $B = 0$, esto es debido a la necesidad de obtener el mayor beneficio a nivel de sistema, que a nivel individual.
2. Las permitidas por la tecnología sin aquellos donde los dispositivos tiene hardware o aplicaciones donde existe una ganancia inicial individual $B > C$ y $C = 0$
3. Las permitidas socialmente son en los cuales el usuario gana reputación en su grafo social, puede obtener beneficio o no, la relación es $B > 0$ y $C > 0$, es decir algunos gana otros invierten.
4. Finalmente el altruismo sigue la regla de Hamilton, modelada por William Hamilton en 1963, los usuarios no ganan demasiado pero se sienten felices de saber que pueden ayudar a otros, la re-

lación es descrita como $B \cdot r > C$, donde r es la relación entre las dos entidades, esta relación es valida si $r > 1$.

A.3. Internet de las Cosas (IoT)

La inclusión de comunicaciones en sistemas informáticos ha permitido nuevas dinámicas entre dispositivos y usuarios, implementar nuevos servicios y una nueva demanda de características en los sistemas informáticos. La movilidad y el control de recursos son elementos clave para monitorear y mejorar los sistemas computacionales actuales; la evolución de hardware y comunicación brindan un nuevo conjunto de tecnologías para explotar sistemas tradicionales como Internet, Educación, Vigilancia, Redes entre otros.Nan y cols. (2018); Skarlat y cols. (2017); Shahzadi y cols. (2017)

Hoy en día en los dispositivos móviles existen más tecnologías de red integradas, para clasificar las redes inalámbricas podemos dividirlas en redes de largo alcance, por ejemplo las tecnologías para redes celulares 2G(GSM, CSD, GPRS), 3G (UMTS / HSDPA), 4G como WiMAX, LTE, LTE-A (HSPA + LTE), en este tipo de arquitectura se tiene una estación base central y un conjunto de celdas para proporcionar servicios y cobertura. La segunda forma son las comunicaciones de corto alcance, usadas primero para reducir el tamaño de los dispositivos y explotar la capacidad de computación de los microchips, esto permite crear redes superpuestas con el mismo dispositivo. En esta evolución, destacan dos tecnologías Bluetooth y Wifi (IEEE802.11.x.)

En ambos casos la evolución está relacionada con las velocidades de datos y el ancho de banda, en los teléfonos inteligentes es más común utilizar ambas tecnologías, pero existen cada vez más dispositivos con estas capacidades, en este contexto las nubes móviles Fitzek y Katz (2013), como plataforma para explotar recursos compartidos en un clúster, es un modelo útil para MANETs gracias a la capacidad de configurarse de forma autónoma, por tanto, no hay un control centralizado y pueden auto recuperarse en caso de falla en la topología, perdidas de conexión , energía o cualquier otro evento que cambie la estructura de la red.

Además, los enlaces entre los nodos son temporales ya que se mueven continuamente, esto puede causar inestabilidad. Para una MANET, la escalabilidad puede ser un problema, a medida que la red crece su rendimiento no puede disminuir y debe mantener niveles aceptables de calidad para los servicios ofrecidos. Como los nodos de la red no tienen un suministro continuo de energía y dependen de sus baterías, es necesario tener un modelo de consumo robusto de energía Akhtar y Wang (2016) para que cada nodo haga un uso adecuado de la energía restante y de todos los recursos en la MANET

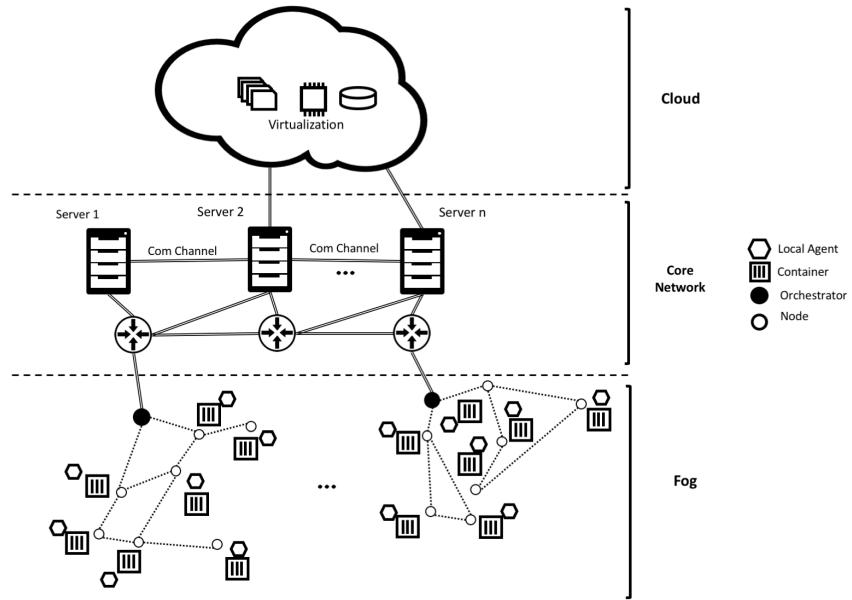


Figura A-3: Arquitectura IoT

Sarkar y cols. (2007).

Este conjunto de características y necesidades, busca incluir mecanismos para explotar comportamientos de autonomía y desplegar sistemas inteligentes en el entorno informático, por ejemplo, los dispositivos de sensores de bajo costo, la infraestructura de computación ubicua y la comunicación inalámbrica, son el núcleo del ecosistema de IoT, es esta relación entre el mundo físico y el mundo digital que ha generado varios avances para planificar el diseño e implementar aplicaciones para ciudades inteligentes, ciudades sostenibles, telemedicina, aplicaciones móviles, que en conjunto constituyen el denominado espacio CyberfísicoBibri (2017).

Internet de las Cosas IoT

Definición: El concepto Internet de las cosas (IoT) Huang y Li (2010), surgió alrededor del año 2010, como la integración del mundo físico con el mundo de la información, las cosas en este contexto son una gran cantidad de sensores, dispositivos integrados, objetos físicos y virtuales, los sistemas inteligentes conectados con los humanos a través de Internet, estos dispositivos están conectados trabajando con el Protocolo de Internet IPv6, en algunos casos funcionan con el IPv4 para operar Bibri (2017), esta arquitectura permite desplegar nuevos servicios que involucran personas, dispositivos, redes e interacciones humano-máquina. Uno de los objetivos principales en este campo es el uso de redes o redes de sensores inteligentes creando un entorno sólido para mejorar el monitoreo y el proceso de decisión en Smart Cities, Sustainable Cities, Smart Farming, Smart Buildings. Una arquitectura tradicional se puede ver en la figura A-3

Los dispositivos y sistemas que operan en los extremos de la red, son denominados dispositivos de borde, esta capa se puede clasificar en tres tipos diferentes Dolui y Datta (2017), Mobile Edge Computing (MEC), Fog Computing (FC) y Cloudlet Computing (CC). MEC incluye interacciones con redes celulares que ofrecen algunos servicios en la nube celular, FC presenta una capa informática antes de la nube para almacenar y procesar datos, finalmente CC se implementa en dispositivos dedicados con más capacidades computacionales, en algunos casos llamados microcentros de datos.

Debido a las redes interconectadas, el uso de diferentes redes y medios para vincular dispositivos y servicios, es necesario vincular las redes convergentes, en algunos casos incluyen la movilidad como una característica de los sistemas con el fin de reducir la latencia y mejorar el rendimiento, adicionando la gestión necesaria de los flujos de trabajo y las interacciones entre las nubes y los dispositivos de usuario, es en este contexto que aparece el concepto Edge Clouds Shahzadi y cols. (2017) para soportar una distribución de la carga en todo el sistema, este modelo propone la mejora de la Calidad de Experiencia (QoE), en las aplicaciones del sistema.

Para resolver algunos problemas con la latencia y la carga computacional entre dispositivos y la nube en un sistema distribuido, es necesario desplegar una arquitectura con nodos o dispositivos heterogéneos y extender las características de dispositivos de nube a borde, en este sentido la arquitectura tiene un conjunto de recursos para explotar y administrar evitando los cuellos de botella de ancho de banda de red, ya que los datos se consumen para los dispositivos de borde, este es el objetivo principal de Fog Computing Hu y cols. (2017). En este orden, los dispositivos de borde pueden proporcionar computación elástica, estando ubicuamente conectados y compartiendo recursos, en los casos principales colaborativamente.

La computación de niebla es la relación entre los dispositivos de borde y el núcleo de la red , Bonomi y cols. (2012) - Bonomi y cols. (2014) o la extensión de la computación en la nube sobre dispositivos de borde, en este sistema existen los mismos recursos (redes, computación y almacenamiento), y comparten en más casos las mismas abstracciones lógicas (Virtualización y Multi-tenancy). El caso típico de uso de la niebla son las aplicaciones con baja latencia, aplicaciones geo - distribuidas (redes de sensores, sistemas de monitoreo) y de gran escala Sistemas distribuidos de control (Smart Grid, Smart Building, Smart Farming). En este entorno Fog, los nodos perciben los recursos como dedicados, sin embargo, es el producto del uso compartido de recursos en la niebla, utilizan los sistemas de archivos virtualizados, infraestructura de virtualización de red (por ejemplo, software definido Redes (SDN)) Negash y cols. (2018). Aún no es clara la clasificación de los dispositivos de borde en computación, sus dominios, arquitectura y servicios, una taxonomía posible se describe en Mahmud y cols. (2018), esta taxonomía propone un conjunto de características para implementar servicios y aplicaciones en un entorno Fog, configuración de nodos, colaboración nodal , métricas de provisión de recursos / servicios, objetivos de nivel de servicio, sistema de red aplicable y seguridad. En este documento, es un factor clave las métricas de aprovisionamiento en recursos y aplicaciones y la administración del sistema de red, para satisfacer y desplegar los servicios en la niebla.

En el mismo contexto, una arquitectura propuesta para implementar servicios y compartir recursos se muestra en Brogi y cols. (2017) y Brogi y Forti (2017), en este modelo el objeto principal es la colonia de niebla, la cual es un conjunto de recursos y aplicaciones heterogéneos, que se puede optimizar en términos de atributos de calidad de servicio. Para resolver este problema, los autores propusieron un algoritmo genético para reducir el retraso y seleccionar la cantidad correcta de recursos en la colonia de niebla, este modelo fue simulado con iFogSim, para validar el modelo y hacer una prueba predefinida.

En Sood (2018) se introduce un enfoque basado en infraestructura como servicio para niebla y computación en la Nube, se propone un administrador de grupo de recursos para detectar y resolver el problema del interbloqueo, este enfoque es útil para gestionar recursos en el ecosistema de niebla, generando un grupo de recursos llamados de espacio libre, estos deben evitar los bloqueos y asignar los recursos disponibles en la nube pública ó en nubes privadas, la técnica principal usada es el Análisis de redes sociales (SNA), básicamente presenta un grafo con la información sobre la asignación de recursos en la capa de niebla, calcula el tamaño del nodo, al igual que la cantidad de recursos necesarios para un tarea, y efectúa el monitoreo de la cantidad de recursos utilizados para este trabajo. Este desarrollo se implementó en NetLogo 5.1.

Otro Framework desarrollado para Fog computing Nan y cols. (2018) propone un modelo con tres niveles: nivel de cosas, nivel de nube y nivel de niebla; en el primer nivel las cosas se implementan

en los sensores inalámbricos, redes de actuadores y dispositivos móviles, estos dispositivos envían información al nivel de niebla, en los nodos de niebla (conmutadores y enrutadores) y vinculan el sistema inalámbrico con el nivel de nube mediante enlaces alámbricos. El enfoque incluye más cálculos para las aplicaciones complejas en la niebla y despliega servicios que un nodo Fog no puede implementar con sus recursos. En este framework, los autores presentan un algoritmo en línea, llamado optimización de ranura unitaria, basado en la técnica de optimización de Lyapunov para equilibrar la compensación de tres vías entre el tiempo promedio de respuesta, el costo promedio y el número promedio de pérdida de la aplicación.

En algunos casos, el elemento más interesante en una arquitectura de computación de niebla es un Orquestador Santos y cols. (2017), el desarrollo de este tipo de artefacto computacional agrega un nuevo conjunto de políticas para administrar los nodos de niebla y la interacción con los nodos de la nube. Para el desarrollo de los orquestadores existen lineamientos como las directrices del Instituto Europeo de Normas de Telecomunicaciones (ETSI), la arquitectura de las funciones de virtualización de red (NFV) y la arquitectura de orquestación (MANO), estos lineamientos representan una base para la gestión y orquestación de aplicaciones en contextos como ciudades inteligentes, el Fog Orchestrator (FO) es responsable de la gestión del ciclo de los microservicios en el sistema, el monitoreo y análisis, y el responsable de la gestión de la interfaz que contiene el modelo de decisión de niebla registros y servicios de red.

Otro campo para implementar y explotar la arquitectura de Fog Computing son los Cyber Physical Systems (CPS) Lee y cols. (2015), se considera como "sistema de sistemas" donde un conjunto de sistemas heterogéneos interactúan de manera continua, y juntos diseñan la arquitectura general de CPS Khaitan y McCalley (2015). Algunas características de CPS son alto grado de automatización, creación de redes, reorganización/reconfiguración dinámica y capacidad cibernetica en cada componente físico, con el fin de mejorar las interacciones entre los sistemas y los protocolos de red en los diferentes niveles de operación, una aplicación realizada simulando redes ad hoc se describe en Li y Negi (2011), para gestionar las células eléctricas de un automóvil eléctrico con dispositivos inalámbricos, el enfoque es un planificador para optimizar la transmisión de paquetes y el método utilizado es una combinación de optimización de Lyapunov y técnicas de muestreo de cadenas de Markov (Markov Chain Monte Carlo -MCMC).

Otra aplicación interesante con CP se describe en Gu y cols. (2017), orientada a sistemas médicos ciberfísicos (MCPS), incluye una red y un modelo para reducir el costo de implementación de servicios médicos virtuales en redes celulares, el objetivo principal es mejorar la ejecución de aplicaciones en dispositivos médicos, donde la estación base proporciona no solo los servicios de comunicación a los dispositivos médicos, sino también los recursos de computación y almacenamiento para alojar las

aplicaciones de dispositivos médicos virtuales (VMD) como máquinas virtuales (VM).

A.4. Protocolos de Enrutamiento en MANETS

Un elemento fundamental en el funcionamiento de las redes ad hoc, vital para exhibir las capacidades de auto-configuración, tolerancia a comportamientos dinámicos, son los protocolos de enrutamiento, estos algoritmos le permiten a la red ad hoc encontrar los nodos vecinos, las rutas entre dispositivos y mantener la disponibilidad de servicios dentro de la red.

La necesidad de mitigar los cambios de topología debido a la movilidad en MANETS al igual que la incertidumbre implícita del medio inalámbrico, los problemas asociados al terminal oculto y al terminal expuesto, hacen una fuente de estudio los protocolos de enrutamiento en MANETS, sin contar los problemas asociados a las restricciones de recursos, no sólo de cómputo sino también de espectro, ancho de banda y energía Loo y cols. (2016).

Estos protocolos han evolucionado a lo largo del tiempo han sido clasificados en tres grandes grupos los reactivos, proactivos e híbridos. Los protocolos reactivos o por demanda como los protocolos de Vector Distancia AODV (Ad hoc on demand distance Vector), DSR (Dynamic Source Routing) hacen peticiones de la tabla de enrutamiento cuando es requerido enviar algún mensaje o petición, en algunos casos los hace lentos pero contribuyen al ahorro de energía, al solicitar las rutas cuando las requieren, generan latencia al momento de encontrar el mejor camino, e incluso el problema de cambio de topología en ambientes altamente móviles, por su parte los protocolos pro-activos de estado de enlace los cuales periódicamente realizan publicación de rutas, como el protocolo OLSR (Optimized Link State Route) o el protocolo BATMAN (Better Approach to Mobile Ad hoc Networking), mediante inundaciones de paquetes tipo *hola* a lo largo de la red manteniendo las tablas de enrutamiento actualizadas, mejorando el descubrimiento de vecinos y la publicación de rutas Fan y cols. (2016); Mouradian y cols. (2016). Finalmente están los protocolos híbridos que utilizan elementos combinados de los dos primeros enfoques.

Destino	Siguiente Salto	Distancia	Interfaz
3	2	2	a1
4	2	2	a1
6	5	2	a2
8	5	2	a2

Tabla A-1: Modelo de Tabla en Protocolos Reactivos Loo y cols. (2016)

Descubrimiento de Rutas

El descubrimiento de rutas es un proceso desencadenado por la necesidad de una fuente para alcanzar o enviar información a un destino, la ruta es mantenida por el proceso de mantenimiento de rutas hasta que la ruta no es requerida de nuevo. De esta manera la sobrecarga de comunicación es reducida al igual que el consumo de energía, esto es en Protocolos por demanda o reactivos.

Los protocolos proactivos en cambio mantienen la ruta actualizada como se ve en la tabla A-1, este intercambio de información periódica es bueno cuando la estructura de la red cambia de forma constante, la revisión de la tabla de enrutamiento es más rápida en ambientes con baja movilidad y requiere menos energía que buscar en toda la red un destino.

En la figura A-4 se presenta una corta taxonomía de los protocolos de enrutamiento en MANETs.

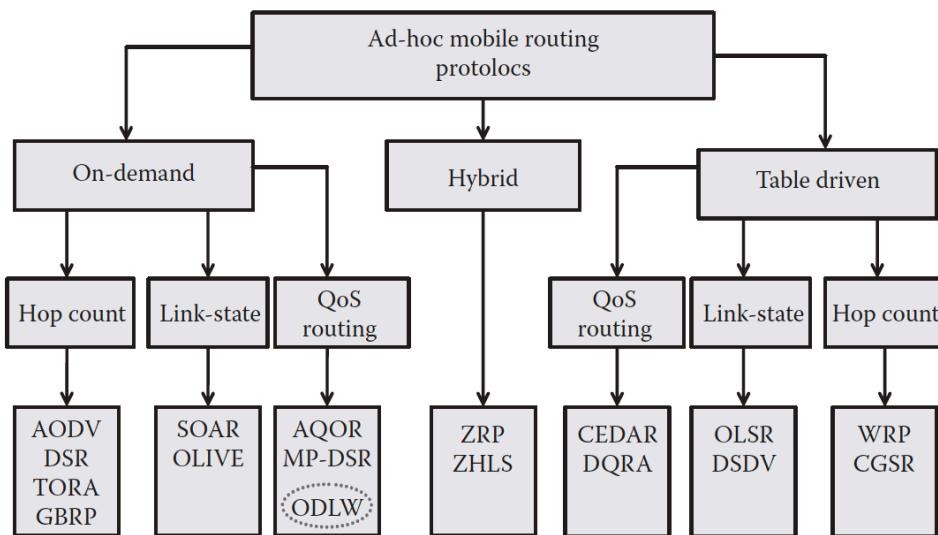


Figura A-4: Protocolos de Enrutamiento en Manets

A.4.1. Protocolo B.A.T.M.A.N

Este protocolo fue desarrollado en Alemania por la comunidad Freifunk (<http://freifunk.net/en/>), la cual es una iniciativa no comercial para redes inalámbricas de uso libre Neumann y cols. (2008). Basados en su experiencia con su propia implementación del protocolo OLSR, concluyeron que se necesitaba un nuevo protocolo de enrutamiento. La razón principal para esta decisión era el tamaño de la red con más de 400 nodos participantes.

El principio de funcionamiento de este protocolo de enrutamiento se puede resumir de la siguiente manera:

1. Después de un intervalo de tiempo dado cada nodo debe emitir para todos los demás (broadcast) un mensaje originador (OGM), que básicamente establece la existencia del nodo.
2. Para que los nodos que están fuera del alcance del nodo originador sepan de su existencia, los mensajes OGM son re-emitidos por los nodos receptores de acuerdo con ciertas reglas:
 - El nodo que recibe el mensaje memoriza el vecino directo a través del cual recibió el OGM. Usando las estadísticas de la llegada de mensajes exitosos un nodo puede concluir cuál de sus vecinos directos es el mejor para reenviar los paquetes hacia su destino.
 - Las rutas y la topología de la red sólo es conocida por los nodos que están dentro del alcance directo.
3. La información de enrutamiento se distribuye en toda la red. Esto es suficiente ya que los nodos participantes sólo tienen influencia sobre la decisión del siguiente salto. Estas interacciones se pueden observar en la figura A-5

A.4.1.1. Tipos de Páquetes B.A.T.M.A.N

Dentro del protocolo B.A.T.M.A.N existe un esquema de señalización de las tramas y de los paquetes generados para el descubrimiento de rutas, servicios como seguridad, network coding, unicast , multicast y banderas adicionales para indicar los estados de los paquetes sobre la red, esta información puede ser utilizada para predecir el comportamiento de la red y diferenciar tráfico de control y de datos. Los paquetes principales son los siguientes:

- $@BATADV_I V_{OGM}$: mensaje originador (OGM) para B.A.T.M.A.N. Advance versión IV.
- $@BATADV_BCAST$: paquetes que transportan datos con destino broadcast.

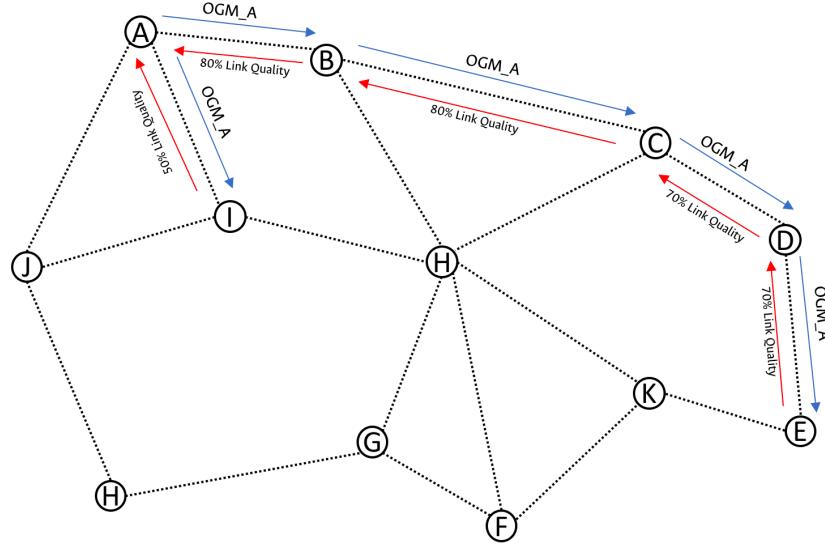


Figura A-5: Protocolo Batman

- @ $BATADV_{CODED}$: paquetes codificados con network coding.
- @ $BATADV_{UNICAST}$: paquetes que transportan datos con destino unicast.
- @ $BATADV_{UNICAST_FRAG}$: paquetes unicast que transporta una parte (fragmento) de los datos del paquete original.
- @ $BATADV_{UNICAST_4ADDR}$: paquetes unicast que incluyen la dirección origen del nodo emisor.
- @ $BATADV_{ICMP}$: paquetes unicast similares a ICMP utilizados para realizar ping o traceroute.
- @ $BATADV_{UNICAST_TVLV}$: paquetes unicast que transportan contenedores TVLV.

PROTOCOLO BATMAN

Observación: Para el desarrollo e implementación del modelo de virtualización inalámbrica de este trabajo se usó el protocolo BATMAN version 2017-2 y los servicios de ALFRED para la mensajería sobre una red Ad hoc, y el conjunto de comandos asociados del paquete Batctl 2017-2

Apéndice B

Anexo: Modelamiento Algebraico de Sistemas Distribuidos

Los modelos de cómputo están compuestos en su mayoría por tres características comunes de los lenguajes de programación, *la sintaxis, los tipos y la semántica*, con base en estos elementos se pueden clasificar los modelos de cómputo en cuatro grupos fundamentales Bruni y Montanari (2017), para describir las interacciones , su estructura, los tipos y los niveles de representación necesarios para describir un modelo de cómputo, los cuatro grandes grupos son:

- **Modelos Imperativos -(IMP):** , en estos modelos se introducen la notación λ , la recursion , la equivalencia programada, la composicionalidad, la integridad y la completitud.
- **Modelos funcionales de Alto nivel -(HOFL):** en estos se estudian los roles de los tipos de sistemas, los conceptos principales de este tipo de modelos provienen del dominio teórico y las distinciones de evaluación perezosa y ansiosa.
- **Modelos Concurrente no deterministas (CCS):**Estos modelos están basados en operaciones semánticas etiquetadas, se introduce la notación de bisimulación, equivalencias y congruencias observacionales. Modelos de movilidad de nombres y especificaciones de sistemas con lógica modal y temporal.
- **Modelos Probabilísticos - Estocásticos (PEPA):** son modelos donde se explotan las teorías de las cadenas de Markov, para realizar análisis cuantitativos de sistemas posiblemente concurrentes.

Teniendo en cuenta estos tipos de modelos de cómputo, es necesario indicar su relación con modelos matemáticos, estos permiten a su vez entender y razonar sobre el comportamiento de los progra-

mas, habilidad necesaria para interpretar completamente el significado de las construcciones de los programas, identificando como analizarlos y verificar con precisión su comportamiento.

Inicialmente en esta sección se hará un recuento de la historia de las álgebras de procesos, algunos cálculos particulares, para finalmente introducir el cálculo π como modelo de representación del sistema distribuido propuesto en este trabajo

B.1. Cálculo II

El cálculo π es un modelo formal para la especificación, análisis y verificación de sistemas compuestos por procesos concurrentes que se comunican. Varela y Agha (2013) El cálculo π ofrece un marco conceptual para entender la movilidad, al igual que un conjunto de herramientas matemáticas para representar sistemas móviles y razonar sobre su comportamiento. Sangiorgi y Walker (2003)

El cálculo π es de la familia del álgebra de procesos, esta Familia está compuesta por diversos tipos como los son CCS - (Calculus of Communication Systems), CPS (communicating sequential processes), ACP (Algebra of Communicating Processes), entre otros. Algunas variaciones del cálculo π son : Distributed Join Calculus, Distributed pi-calculus, Ambient calculus y Oz son lenguajes influenciados por el cálculo π .

El cálculo π tiene dos aspectos, el primero de ellos es ser teoría de sistemas móviles y el segundo ser un modelo general de computación. Como teoría de sistemas móviles, ofrece una rica mezcla de técnicas para razonar sobre el comportamiento de los sistemas. Como modelo de computación toma la **interacción como primitiva**.

En el cálculo λ la primitiva es la **aplicación** de la función. De forma similar como el cálculo λ es la base de los lenguajes funcionales, el cálculo π - y las variantes- son la base de varios lenguajes de programación concurrentes como Pict, Join y TyCO, su unidad funcional es el proceso, Un **proceso** es una serie de acciones o eventos, nuestro caso un **proceso** es el comportamiento de un sistema.

B.1.1. Definiciones

Recordando que el cálculo π ofrece un marco conceptual para entender la movilidad junto con las herramientas matemáticas para representar sistemas móviles y razonar sobre su comportamiento. Surgen preguntas como: ¿Qué es movilidad? Cuando se habla de sistemas móviles, ¿Cuáles son las en-

tidades que se mueven dichas entidades? ¿En qué espacio se mueven. Una respuesta a estas preguntas se puede dar considerando dos tipos de movilidad.

Movilidad

Existen dos tipos de movilidad de **Enlaces** que se mueven en un espacio abstracto de procesos enlazados y **Procesos** que se mueven en un espacio abstracto de nombres enlazados.

El cálculo π trata la primera clase de movilidad: expresa de forma directa el movimiento de enlaces en un espacio de procesos enlazados, en este sentido hay dos clases de entidades básicas en el cálculo π : nombres y procesos. La computación concurrente es modelada en el cálculo π como procesos que se comunican sobre canales compartidos, para tener un acercamiento inicial con esta forma de representación se presentan las siguientes consideraciones:

- Las variables de Procesos son representadas con letras Mayúsculas P, Q, R .
- Los canales y nombres son denotados con letras minúsculas a, b, c .
- Las interacciones sobre los canales (Lectura y Escritura), se representan como el prefijo α

Para tener una idea más precisa de la sintaxis y del modelado de procesos usando Cálculo π , revisaremos un ejemplo sencillo donde se tiene un modelo cliente servidor como se ve en la figura B-1, para imprimir el contenido de un archivo. Milner (1999), en contraparte se puede observar su representación sobre cálculo π en la figura B-2.

Los siguientes son los elementos que componen el ejemplo y su respectiva notación en cálculo π .

- P es el proceso cliente.
- S es el proceso Servidor .
- τ es el operador de transformación .
- a es el acceso a la impresora .
- b es el acceso al servidor.
- c es un marcador de posición.
- d es el acceso a algún dato.

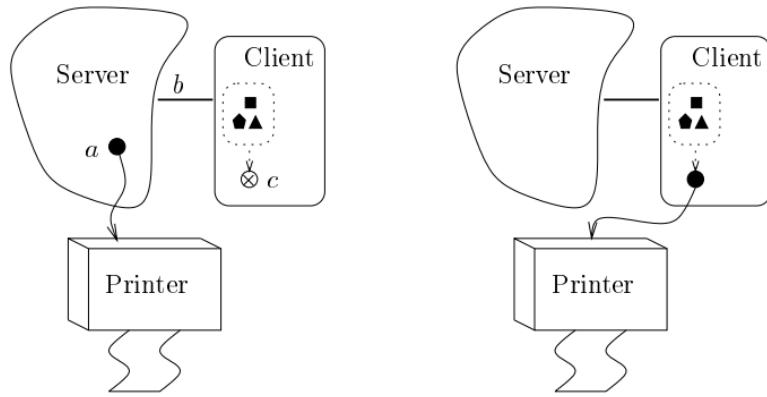


Figura B-1: Servidor de impresión (Izq. antes de la interacción, Der. Después de la interacción) $(\bar{b}a.S|b(c).\bar{c}d.P) \xrightarrow{\tau} (S|\bar{a}d.P)$ Milner (1999)

- $(\bar{b}a.S|b(c).\bar{c}d.P) \xrightarrow{\tau} (S|\bar{a}d.P)$ Representación de las interacciones de los procesos cliente y servidor .
- $(\nu.a)(\bar{b}a.S|R)|b(c).\bar{c}d.P \xrightarrow{\tau} (\nu.a)(S|R|\bar{a}d.P)$ Representación de las interacciones denotando los puertos ν .

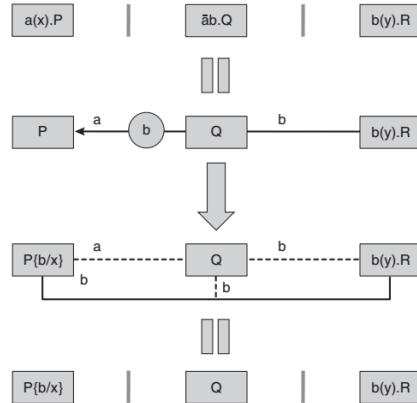


Figura B-2: Modelo cliente servidor Cálculo II Varela y Agha (2013)

B.1.2. Semántica operacional

El elemento básico de operación e cálculo π es el nombre \mathcal{N} denotado como un conjunto infinito con un rango a, b, c, \dots, z elementos los cuales podrían funcionar como puertos de comunicación,

variables y datos de las variables, un conjunto de (agentes) identificadores sobre \mathcal{A} . Los elementos mas relevantes para realizar operaciones son:

- El agente vacío **0** el cual no desempeña ninguna acción.
- El *prefijo de Salida* $\bar{a}x.P$. el nombre x es enviado a lo largo del nombre a , y a partir de entonces el agente continua como P .
- El *prefijo de Entrada* $a(x).P$. indica que un nombre es recibido a lo largo de a y x es un lugar para recibirlo, Después el agente de entrada continua como P , pero con el nombre recién recibido reemplazando a x .
- El *prefijo de Silencio- no interacción* $\tau.P$. el cual representa a un agente que puede evolucionar a P sin interacción con el ambiente.
- Una *Suma* $P + Q$ indica un agente que puede representar cualquier P . o Q
- Una *Composición Paralela* $P|Q$ representa el comportamiento combinado de P y Q ejecutandose en paralelo.
- Un *Match if* $x = y$ then P . se espera que este agente tenga el mismo comportamiento de P si x y y tienen el mismo nombre, de lo contrario no hace nada.
- Un *Mismatch if* $x \neq y$ then P . este agente podría comportarse como P si x y y no tienen el mismo nombre, de lo contrario no hace nada.
- Una *Restricción* $(\nu x)P$. este agente se comporta como P pero este agente es local, significa que este inmediatamente no puede ser usado como un puerto para comunicarse entre P y su ambiente.
- Un *Identificador* $A(y_1, \dots, y_n)$ donde n es la aridad de A cada identificador tiene una *Definición* $A(x_1, \dots, x_n) \stackrel{\text{def}}{=} P$ donde x_i puede ser una pareja diferente y la intuición es que $A(y_1, \dots, y_n)$ se comporta como P con y_i reemplazando a x_i por cada i . Puede verse como la declaración de un proceso x_1, \dots, x_n como parámetros formales y el identificador $A(x_1, \dots, x_n)$ como una invocación con los parámetros actuales de y_1, \dots, y_n

Aunque la representación del cálculo π permite validar la concurrencia dentro de un sistema, modelar las interacciones entre procesos, la movilidad de los mismos, presenta restricciones para modelar interacciones y cambios de estados en sistemas ubicuos donde se requiere de una representación espacial y lógica, es por ello que se continua este estudio con el modelo de Bigrafos B.2, donde es posible modelar el sistema propuesto en este trabajo.

B.2. Bigrafos

Los sistemas Bigraficos Reactivos Milner (2009, 2008) (BRSs en inglés) son modelos de computación en los cuales la *ubicación* y la *conectividad* son esenciales, en el contexto de IoT e incluso de Internet la ubicación geográfica es un desafío para el control de sistemas distribuidos. Los bigrafos son desarrollados de el cálculo de acciones, e involucra ideas de distintas fuentes como la Máquina Abstracta Química (Cham), de Berry y Boudol, el cálculo π de Milner, Parrow y Walker , presentado en la sección B.1, las redes de interacciones, los ambientes móviles, las fusiones explicitas de Gardner y Wischik, el Nomadic Pict de Wojciechowski y Sewell y la teoría de sistemas reactivos de Leifer y Milner, al igual que una contribución de las redes de Petri.

Este modelo ha sido desarrollados con dos objetivos claros *i*) ser capaz de modelar directamente aspectos de sistemas ubícuos enfocados en la conectividad móvil(grafo de enlace) y la movilidad local (grafo de lugar) simultáneamente, *ii*) proveer una unificación de las teorías existentes para el desarrollar una teoría general en la cual varios modelos de cálculos de sistemas concurrentes y móviles puedan ser representados.

Bigrafo

En esencia un bigrafo consiste de un **un grafo de lugar**; un bosque cuyos nodos representan una diversidad de artefactos computacionales, y un nodo **un grafo de enlace**, el cual es un hypergrafo que conecta los puertos de los nodos. Los bigrafos pueden ser reconfigurados por el significado de las **reglas de reacción**. El bigrafo \breve{G} tiene nodos $V = \{v_0 + \dots + v_5\}$ y vértices $E = \{e_0, e_1, e_2\}$

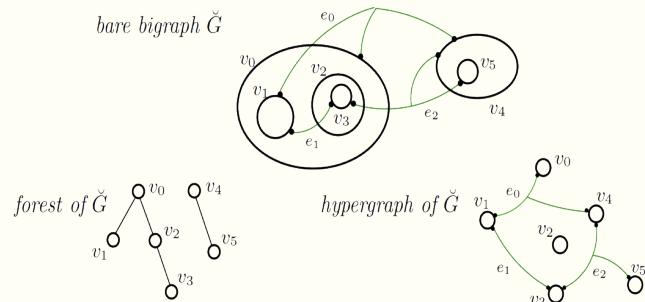


Figura B-3: Bigrafo adaptado de Milner (2008)

Definición: Una regla de reacción consiste de un **redex** (el patrón a ser cambiado) y un **reactum** (el patrón de cambio), los cuales reconfiguran el bigrafo y describen las interacciones de conectividad y lugar de los nodos

Los sistemas ubícuos son modelados en espacios discretos compartidos involucrando tres conceptos: agente , ubicación y conectividad, cuando se reconfigura el espacio hay dos conceptos adicionales el movimiento y la interacción, del mismo modo el movimiento es representado por reglas de reacción

las cuales modifican el espacio y la conectividad, es decir hay un cambio de estados en el bigrafo.

Bigrafo

Un bigrafo con nodos V y vértices E tiene un bosque cuyos nodos son V ; este a su vez tiene un hypergrafo con nodos V y vértices E

B.2.1. Definiciones

Un bigrafo es un par $B = \langle B^P, B^L \rangle$ un grafo de lugar y un grafo de enlace estos son sus constituyentes, su cara externa esta denotada por el par $\langle n, Y \rangle$ donde n y Y son la cara externa de B^P y B^L respectivamente, del mismo modo para la cara interna $\langle m, X \rangle$.

Cada Bigrafo G tiene dos interfaces I y J , definidas como $G : I \rightarrow J$, I es la interface interna y J la interface externa. Una interfaz es una tripla $\langle m, \vec{X}, X \rangle$, donde m es la amplitud (número de sitios o raíces), X el conjunto completo de nombres locales y globales, y \vec{X} indican las ubicaciones de cada nombre local.

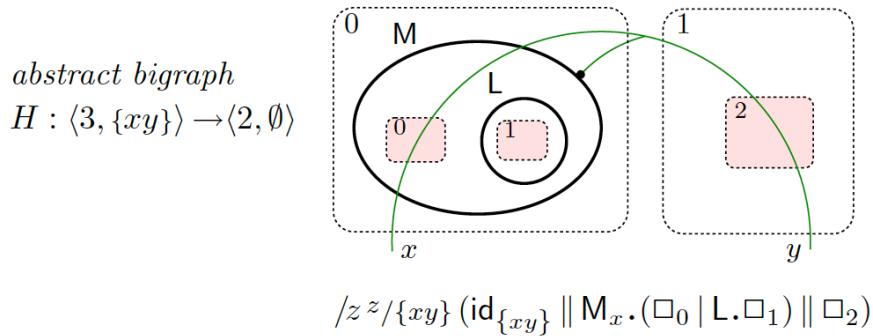


Figura B-4: Bigrafo Abstracto Milner (2008)

Un bigrafo abstracto H como el mostrado en la figura B-4, es aquel que no tiene denotados los nombres de los nodos, ni de los enlaces. Los nodos de un bigrafo puede ser de diferentes tipos; esto refleja que pueden contribuir diferenciadamente a las dinámicas. Cada bigrafo tiene asignado un tipo llamado control K el cual determina la aridad libre ligada por $v : K$, este control indica la forma de acceder

al nodo, aplicación u otro elemento del bigrafo, estos son especificados en un *firma*

$$K = \{K : 2, L : 0, M : 1\} \quad (\text{B-1})$$

La firma del bigrafo G de la figura B-5 esta dada en la ecuación B-1, la cual representa la aridad de cada uno de los nodos.

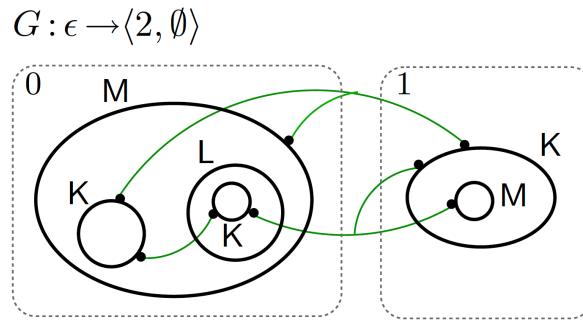


Figura B-5: Bigrafo con controles Milner (2008)

Construcción

Se pueden hacer bifragos más grandes a partir de unos más pequeños a través de sus interfaces; esta construcción es definida en términos de los grafos constituyentes de lugar y enlace

Definición B.2.1 (bigrafo concreto puro) Un bigrafo puro (concreto) sobre una firma \mathcal{K} , toma la forma $G = (V, E, ctrl, G^P, G^L) : I \rightarrow J$ donde $I = \langle m, X \rangle$ y $J = \langle n, Y \rangle$ son sus caras internas y externas, cada una combinando una amplitud con un conjunto finito de nombres globales plasmados sobre X . Esto dos primeros componentes V y E son conjuntos de nodos finitos y vértices respectivamente. El tercer componente $ctrl : V \rightarrow \mathcal{K}$, un mapa de control, asignado a cada nodo. Los restantes dos son:

$$G_P = (V, ctrl, prnt) : m \rightarrow n \text{ un grafo de lugar}$$

$$G_P = (V, E, ctrl, link) : X \rightarrow Y \text{ un grafo de lugar}$$

Definición B.2.2 (grafo de lugar) Un grafo de lugar $A = (V, ctrl, prnt) : m \rightarrow n$ tiene una amplitud interna n y una externa m , ambos ordinales finitos, un conjunto de nodos V con un mapa de control $ctrl : V \rightarrow \mathcal{K}$; y un mapa principal $prnt : m \uplus V \rightarrow V \uplus n$. El mapa principal es acíclico, que es $prnt^k(v) \neq v$ para todo $k > 0$ y $v \in V$.

Definición B.2.3 (grafo de enlace) Un grafo de enlace $A = (V, E, ctrl, link) : X \rightarrow Y$ tiene un conjunto X de nombres internos, Y de nombres externos, V de nodos y E de enlaces. También tiene una función $ctrl : V \rightarrow \mathcal{K}$ llamada mapa de control, y una función $link : X \uplus P \rightarrow E \uplus Y$ llamado mapa de enlace. donde $P \stackrel{\text{def}}{=} \sum_{v \in V} ar(ctrl(v))$ es el conjunto de puertos de A .

B.2.2. Operaciones Básicas

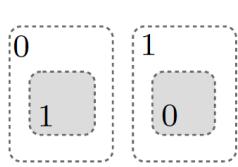
Es de resaltar que todos los bigrafos pueden ser construidos desde otros elementales a partir de la composición (\circ) y el producto tensor \otimes , existen otras dos operaciones derivadas que son el producto paralelo $G_1||G_2$ y el producto principal $G_1|G_2$, estos productos son definidos en las interfaces del bigrafo.(La función identidad es usada en esta operaciones es descrita como Id_s).

Para los bigrafos G_1 y G_2 que no comparten nombres o nombre internos, es posible realizar un producto tensor $G_1 \otimes G_2$ por la yuxtaposición de sus grafos de lugar, construyendo la unión de sus grafos de enlace, e incrementando los indices de los sitios en G_2 por el número de G_1 .

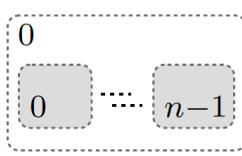
En cambio el producto paralelo $G_1||G_2$ es como el producto tensor, excepto que los nombres globales pueden ser compartidos: if y es compartido, todos los puntos de y en G_1 y G_2 se convierten en puntos de y en $G_1||G_2$.

Se pueden componer bigrafos $G_2 : I \rightarrow I'$ y $G_1 : I' \rightarrow J$, manteniendo el bigrafo $G_1 \circ G_2 : I \rightarrow J$ al conectar los sitios de G_1 con las raíces de G_2 , eliminando ambos y conectando nombres de G_2 con nombre internos de G_1 .

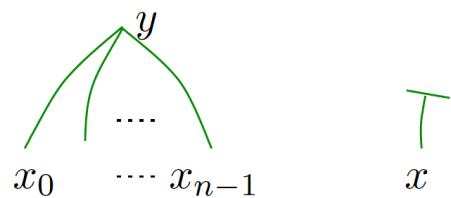
elementary placings and linkings



$swap : 2 \rightarrow 2$



$merge_n : n \rightarrow 1$



$y/X : X \rightarrow y$

$/x : x \rightarrow \epsilon$

Figura B-6: Bigrafos Elementales adaptado de Milner (2009)

Continuando con este acercamiento a la teoría de bigrafos se enunciaran los bigrafos elementales, usados para la construcción de reglas de interacción y las operaciones entre bigrafos,básicamente existen tres tipos de bigrafos elementales, las primeras dos clases son libres de nodos. Si un bigrafo libre de nodos tampoco tiene enlaces se llama colocación; si no tiene lugares se llama un enlace. Hay lugares y enlaces elementales, desde los cuales todos los otros pueden ser formados usando composición y producto, estos se ven en la figura B-6, swap es la operación de intercambio entre regiones, merge es la unión de los lugares, estos dos bigrafos se forman a partir de los grafos de lugar, por parte de los grafos de enlace están el bigrafo de sustitución donde es reemplazado y escrito el nombre en el enlace e interfaz indicados, la closure es el cierre de un x -enlace.

La tercera clase de bigrafo elemental es el ion discreto que es un simple nodo de control K de aridad n con sus puertos enlazados en n diferentes nombres de salida x y tiene simplemente un sitio , como se ve en la figura B-7.

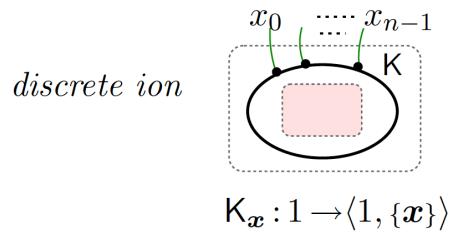
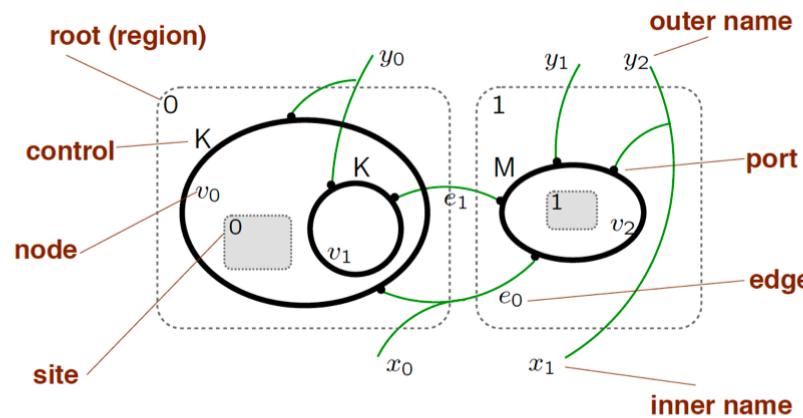


Figura B-7: Ion Discreto Milner (2009)

Varios sistemas han sido modelados con bigrafos como el modelo de comunicación IEEE 802.11 Calder y Sevignani (2014), sistemas multiagente Mansutti y cols. (2014), sistemas de archivos Birkedal y cols. (2007) y por supuesto este trabajo es uno de los sistemas propuestos para ser modelado, en la sección 7 se expondrá el modelo realizado con la rigurosidad matemática más alta posible.

Para terminar esta breve introducción a la teoría de bigrafos la figura B-8 resume y representa los componentes de un bigrafo, estos elementos permiten identificar las reacciones, y en general las abstracciones físicas y lógicas posibles, con el fin de definir las reglas de reacción e identificar las interacciones entre nodos, lugares, raíces, enlaces, nombres internos, nombre externos e interfaces.

The anatomy of bigraphs



place = **root** or **node** or **site**

link = **edge** or **outer name**

point = **port** or **inner name**

Figura B-8: Anatomía de un Bigrafo Milner (2009)

Apéndice C

Anexo:Técnicas de Diseño y Análisis Estructurado- SADT

Para definir los módulos y las relaciones entre las abstracciones propuestas es necesario seleccionar una metodología de diseño que permita enlazar los diferentes niveles y las relaciones entre los mismos dentro de la solución propuesta en este trabajo, para ello se usó una técnica jerárquica para describir con mayor detalle los componentes y elementos claves del sistema virtualizado propuesto.

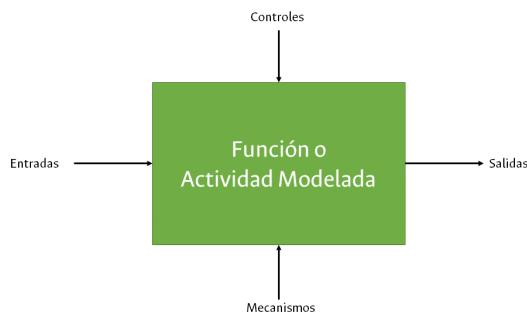


Figura C-1: Diagramas SADT.

La Técnicas de Diseño y Análisis Estructurado - SADT Marca y McGowan (1987) es un lenguaje gráfico para describir sistemas basados en relaciones jerárquicas, como finalidad un sistema modelado con SADT se enfoca en identificar las fase de análisis y diseño de sistemas, maneja un enfoque arriba hacia abajo para describir en bloques funcionales sistemas, posteriormente evolucionó y es conocido dentro de los estándares de IDEF, con la identificación IDEF0 continuando con las bases de la técnica SADT como herramienta en la ingeniería de software.

Modelo SADT

UEsta técnica cuenta con cuatro elementos para definir las funciones o actividades a modelar, como se ve en la figura C-1, la descripción de cada uno de ellos es la siguiente:

- **Controles:** Son las elementos que agregan restricciones a las actividades
- **Entradas:** Son los elementos que son transformados por las actividades
- **Salidas:** Son las entradas transformadas
- **Mecanismos:** Son los elementos que llevan a cabo la actividad

Para tener una vista mas detallada de la virtualización se presentan los diagramas detallados de cada uno de los elementos de este trabajo usando SADT como herramienta descriptiva de los módulos desarrollados. Para el desarrollo de este trabajo se crearon las formas descrita sobre la figura C-2, basadas en esta técnica y generalizar la descripción acorde con los objetivos de este trabajo.

Las formas tienen jerarquías, implican procesos determinísticos, estocásticos y difusos, para describir las interacciones y la especificación de cada uno de los niveles donde se ha descrito el diseño de la solución propuesta.

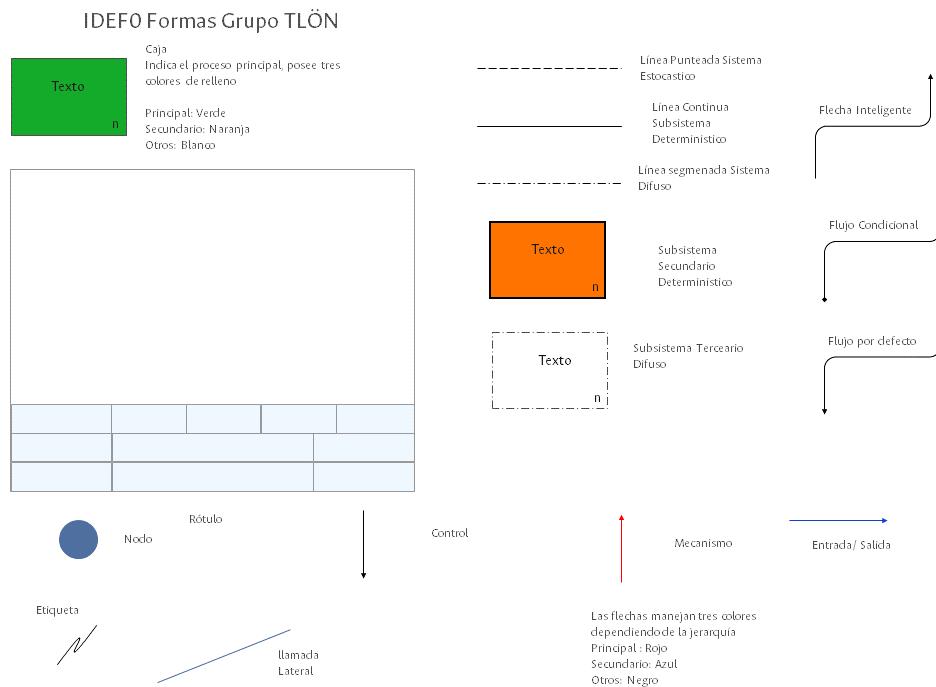


Figura C-2: Formas SADT Tlon.

Apéndice D

Anexo: Instalación de la Solución

```
#####
# Ad hoc System Monitoring Docker Container Apps -Orchestrator  #
#####
```

Introduction

=====

This Framework, has an objective check and monitoring the resources of devices linked over and ad hoc network, this devices provides services of monitoring, sensoring and work as IoT devices, those apps are working on a Docker containers and have the possibility of change de Quota, cpu, memory and others resources, this software is compose for an Orchestrator, that have policies to deploy the apps, this computational artifact is linked with Local Agents on each node, this communication allow to Orchestrator knows the node and application throughput and performance.

All this framework works with B.A.T.M.A.N protocol, A.L.F.R.E.D services and some programs in C++ and Python.

How does it work ?

=====

This framework has four components:

1)The controller:

2) The App:

3) The Local Agent

4) The Orchestrator

Inside an docker containers, there is a controller checks the cpu, this software have a software uses a Local agent, this agnet work together with a local control energy model for s set of devices as Raspberry pi zero, raspberry pi 2, raspberry pi 3 and Odroid UX3.

Install Dependencies

You need run the setup.py file to install all packages, check the A.L.F.R.E.D, Batctl and Batman-adv packages versions (2017-2). You need install Docker on your rpi or IoT device, for raspberry pi you can run the setup an set the KERNELPATH, for install all dependencies.

Check Dependencies

BATAMAN-ADV

The batman-adv module is shipped as part of the Linux kernel and as external module. The external module allows to get new features without upgrading to a newer kernel version and to get batman-adv specific bugfixes for kernels that are not supported anymore. It compiles against and should work with Linux 3.2-4.13. Supporting older versions is not planned, but it's probably easy to backport it.

MANUAL COMPILE

Download the modules Alfred, Batctl

and Batman-adv packages versions (2017-2).

To compile against your currently installed kernel, just type:

```
# make
```

if you want to compile against some other kernel, use:

```
# make KERNELPATH=/path/to/kernel
```

if you want to install this module:

```
# sudo make install
```

CONFIGURATION BATMAN-ADV

The batman-adv module is shipped as part of the Linux kernel and as external module. The external module allows to get new features without upgrading to a newer kernel version and to get batman-adv specific bugfixes for kernels that are not supported anymore. It compiles against and should work with Linux 3.2 - 4.13. Supporting older versions is not planned, but it's probably easy to backport it. If you work on a backport, feel free to contact us. :-)

COMPILE

To compile against your currently installed kernel, just type:

```
# make
```

if you want to compile against some other kernel, use:

```
# make KERNELPATH=/path/to/kernel
```

if you want to install this module:

```
# sudo make install
```

The in-kernel module can be configured through menuconfig. When compiling outside of the kernel tree, it is necessary to configure it using the make options. Each option can be set to to y (enabled), n (disabled) or m (build as module). Available options and their possible values are (default marked with an "*")

```
* CONFIG_BATMAN_ADV_DEBUGFS=[y*|n] (B.A.T.M.A.N. debugfs entries)
* CONFIG_BATMAN_ADV_DEBUG=[y|n*] (B.A.T.M.A.N. debugging)
* CONFIG_BATMAN_ADV_BLA=[y*|n] (B.A.T.M.A.N. bridge loop avoidance)
* CONFIG_BATMAN_ADV_DAT=[y*|n] (B.A.T.M.A.N. Distributed ARP Table)
* CONFIG_BATMAN_ADV_MCAST=[y*|n] (B.A.T.M.A.N. multicast optimizations)
* CONFIG_BATMAN_ADV_NC=[y|n*] (B.A.T.M.A.N. Network Coding)
* CONFIG_BATMAN_ADV_BATMAN_V=[y|n*] (B.A.T.M.A.N. V routing algorithm)
```

e.g., debugging can be enabled by

```
# make CONFIG_BATMAN_ADV_DEBUG=y
```

DOCKER

CONFIGURATION DOCKER ON RPI

If you dont have install docker exec file setup.py, first set the KERNELPATH for install one stable version of docker for rpi.

MANUAL INSTALL

Run those commands on your rpi for install

```
sudo apt-get remove docker docker-engine docker.io
```

```
sudo apt-get update
```

```
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
```

```
curl \
gnupg2 \
software-properties-common

curl -fsSL https://download.docker.com/linux/$(. /etc/os-release;
echo "$ID")/gpg | sudo apt-key add -

sudo apt-key fingerprint OEBFCD88

echo "deb [arch=armhf] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
$(lsb_release -cs) stable" | \
sudo tee /etc/apt/sources.list.d/docker.list

sudo apt-get update

sudo apt-get install docker-ce

sudo docker run armhf/hello-world
```

DOWNLOAD DOCKER IMAGE ARM-V7

```
#####
#          RUN FRAMEWORK          #
#####
```

ORCHESTRATOR

The orchestrator has two servers and two clients to manage the network. This model allows the use of the A.L.F.R.E.D. service to know the IP and the node members on a ad hoc network, this manages the network and the resources, checks the status on node, and the general state of the system, the policies module allows to set rules around the system.

If you want to do a test you need to check the workload file (.csv) to add more load at network.

The orchestrator contains the policies and overall info of an adhoc

network.

CONFIGURATION

You need use the folder orchestrator and run the file, the path is

```
# ../Orchestrator
```

Check all files

```
# ls ../Orchestrator
```

and yo see the next files and directories:

Utils	Devices_info.py	Log_Orch.py	logging.conf
Orchestrtor.py	Policies.py	README	setup.py
SThread.py	SystemData.py	UDP_Exec.py	workload.csv
Workload.py			

Check your wifi interface with command

```
# ifconfig
```

or

```
# iwconfig
```

Note: The wifi interface would support ad hoc mode, this framework only works in that mode.

USAGE

```
# sudo python Orchestrator.py
```

Note: In order to avoid problems remove all IP addresses previously assigned to interfaces now used by batman advanced, e.g.

```
# ip addr flush dev eth0
```

LOGGING/DEBUGGING

You can check three diferents logs on this framework

```
# cat ../Orchestrator/orchestrator.log
```

You can see all events on orchestrator

```
# cat ../Orchestrator/SystemOrchestrator.log
```

You can see all info of node and app status, with their Ad hoc IP

```
# cat ../Orchestrator/device_info.csv
```

You can see the node info, and architecture, supports modes and ID

If you want more information over network you can check the batman-adv logs, previous you need compile it.

The additional debug output is by default disabled. It can be enabled during run time. Following log_levels are defined:

```
0 - All debug output disabled  
1 - Enable messages related to routing / flooding / broadcasting  
2 - Enable messages related to route added / changed / deleted  
4 - Enable messages related to translation table operations  
8 - Enable messages related to bridge loop avoidance  
16 - Enable messages related to DAT, ARP snooping and parsing  
32 - Enable messages related to network coding  
64 - Enable messages related to multicast  
128 - Enable messages related to throughput meter  
255 - Enable all messages
```

LOCAL AGENT

The other component is the Local Agent is a software that enable a

link with controller on docker container and save the node state on a particular app, this software contains a local client with the Docker controller. It knows the apps state, an exec routines to enable A.L.F.R.E.D. services and known the orchestrator info, this allows send info and check the orchestrator instructions.

For set instructions over Apps run on Docker containers, set a group of commands for stop containers, run containers, deploy images, set qouta and set cpu.

This framework is available on <https://www.tlon.unal.edu.co/>

CONFIGURATION

You need use the folder orchestrator and run the file, the path is

```
# ../Orchestrator
```

Check all files

```
# ls ../Orchestrator
```

and yo see the next files and directories:

Utils	Devices_info.py	Log_Orch.py	logging.conf
Orchestrator.py	Policies.py	README	setup.py
SThread.py	SystemData.py	UDP_Exec.py	workload.csv
Workload.py			

Check your wifi interface with command

```
# ifconfig
```

or

```
# iwconfig
```

Note: The wifi interface would support ad hoc mode, this framework only works in that mode.

USAGE

```
# sudo python LocalPerfAgent.py
```

Note: In order to avoid problems remove all IP addresses previously assigned to interfaces now used by batman advanced, e.g.

```
# ip addr flush dev eth0
```

LOGGING/DEBUGGING

You can check the Loacl Agent logs on this address

```
# cat ../LocalAgent/local_agent.log
```

You can see all events and interactions with the controller, app and the orchestrator.

CONTACT

=====

Please send us comments, experiences, questions, anything :)

WEBPAGE www.tlon.unal.edu.co

You can also contact the Authors:

Henry Zarate <hzaratec@unal.edu.co>

Jorge Eduardo Ortiz <jeortizt@unal.edu.co>

Antonio Miele <antonio.miele@polimi.it>

Cristiana Bolchini <cristiana.bolchini@polimi.it>

Apéndice E

Anexo: Caracterización de Energía Odroid XU4

Frequenza little	Frequenza big	POWER -IDLE
400	400	2297,95
600	400	2311,96
800	400	2324,19
1000	400	2347,4
1200	400	2381,03
1400	400	2415,33

Tabla E-1: Odroid Little sin Wifi Frecuencia 400MHz.

Frequenza little	Frequenza big	POWER -IDLE
400	600	2319,51
600	600	2329,06
800	600	2344,94
1000	600	2369,08
1200	600	2398,58
1400	600	2442,75

Tabla E-2: Odroid Little sin Wifi Frecuencia 600MHz.

Frequenza little	Frequenza big	POWER -IDLE
400	1000	2393,02
600	1000	2404,25
800	1000	2354,11
1000	1000	2394,03
1200	1000	2422,1
1400	1000	2470,65

Tabla E-3: Odroid Little sin Wifi Frecuencia 1000MHz.

Frequenza little	Frequenza big	POWER -IDLE
400	1600	2438,93
600	1600	2452,4
800	1600	2471,45
1000	1600	2490,81
1200	1600	2527,26
1400	1600	2566,51

Tabla E-4: Odroid Little sin Wifi Frecuencia 1600MHz.

Frequenza little	Frequenza big	POWER -IDLE
400	1800	2634,7
600	1800	2645,67
800	1800	2673,32
1000	1800	2697,35
1200	1800	2751,55
1400	1800	2776,26

Tabla E-5: Odroid Little sin Wifi Frecuencia 1800MHz.

Frequenza little	Frequenza big	POWER -IDLE
400	2000	3040,9
600	2000	3040,17
800	2000	3060,81
1000	2000	3096,29
1200	2000	3130,12
1400	2000	3200,08

Tabla E-6: Odroid Little sin Wifi Frecuencia 2000MHz..

Frequenza little	Frequenza big	POWER -IDLE
400	400	2821,02
600	400	2831,74
800	400	2841,28
1000	400	2857,46
1200	400	2891,77
1400	400	2932,15

Tabla E-7: Odroid Little con Wifi Frecuencia 400MHz..

Frequenza little	Frequenza big	POWER -IDLE
400	600	2817,23
600	600	2838,97
800	600	2920,02
1000	600	2881,7
1200	600	2906,55
1400	600	3012,58

Tabla E-8: Odroid Little con Wifi Frecuencia 600MHz..

Frequenza little	Frequenza big	POWER -IDLE
400	1000	2837,6
600	1000	2859,71
800	1000	2905,5
1000	1000	2943,69
1200	1000	2985,7
1400	1000	2990,64

Tabla E-9: Odroid Little con Wifi Frecuencia 1000MHz..

Frequenza little	Frequenza big	POWER -IDLE
400	1600	3091,95
600	1600	3149,2
800	1600	3129,48
1000	1600	3208,07
1200	1600	3238,95
1400	1600	3278,25

Tabla E-10: Odroid Little con Wifi Frecuencia 1600MHz..

Frequenza little	Frequenza big	POWER -IDLE
400	1800	3246,8
600	1800	3311,17
800	1800	3326,41
1000	1800	3343,93
1200	1800	3357,72
1400	1800	3485,93

Tabla E-11: Odroid Little con Wifi Frecuencia 1800MHz..

Frequenza little	Frequenza big	POWER -IDLE
400	2000	3259,64
600	2000	3583,43
800	2000	3576,23
1000	2000	3617,73
1200	2000	3585,01
1400	2000	3664,85

Tabla E-12: Odroid Little con Wifi Frecuencia 2000MHz.

Referencias

1. Agarwal, R., Motwani, D., y cols. (2009). Survey of clustering algorithms for manet. *arXiv preprint arXiv:0912.2303*.
2. Akhtar, A. M., y Wang, X. (2016). Cross-layer designs for energy-efficient wireless ad-hoc networks. En *Energy management in wireless cellular and ad-hoc networks* (pp. 147–168). Springer.
3. Ansel, J., Chan, C., Wong, Y. L., Olszewski, M., Zhao, Q., Edelman, A., y Amarasinghe, S. (2009). *Petabricks: a language and compiler for algorithmic choice* (Vol. 44) (n.º 6). ACM.
4. Arnold, M. B. (1960). Emotion and personality.
5. Asnaghi, A., Ferroni, M., y Santambrogio, M. (2016). Dockercap: A software-level power capping orchestrator for docker containers. En *Computational science and engineering (cse) and ieee intl conference on embedded and ubiquitous computing (euc) and 15th intl symposium on distributed computing and applications for business engineering (dcabes), 2016 ieee intl conference on* (pp. 90–97).
6. Barbeau, M., y Kranakis, E. (2007). *Principles of ad-hoc networking*. John Wiley & Sons.
7. Basagni, S., Conti, M., Giordano, S., y Stojmenovic, I. (2004). *Mobile ad hoc networking*. John Wiley & Sons.
8. Bibri, S. E. (2017). The iot for smart sustainable cities of the future: An analytical framework for sensor-based big data applications for environmental sustainability. *Sustainable Cities and Society*.
9. Birkedal, L., Damgaard, T. C., Glenstrup, A. J., y Milner, R. (2007). Matching of bigraphs. *Electronic Notes in Theoretical Computer Science*, 175(4), 3–19.
10. Bonomi, F., Milito, R., Natarajan, P., y Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. En *Big data and internet of things: A roadmap for smart environments* (pp. 169–186). Springer.
11. Bonomi, F., Milito, R., Zhu, J., y Addepalli, S. (2012). Fog computing and its role in the internet of things. En *Proceedings of the first edition of the mcc workshop on mobile cloud computing* (pp. 13–16).

12. Borges, J. L., y Clemente, J. E. (1956). *Ficciones*. Emecé Buenos Aires.
13. Brogi, A., y Forti, S. (2017). QoS-Aware Deployment of IoT Applications Through the Fog. *IEEE Internet of Things Journal*, 4(5), 1185-1192. doi: 10.1109/JIOT.2017.2701408
14. Brogi, A., Forti, S., y Ibrahim, A. (2017). How to best deploy your fog applications, probably. En *Fog and edge computing (icfec), 2017 ieee 1st international conference on* (pp. 105–114).
15. Bruni, R., y Montanari, U. (2017). *Models of computation*. Springer.
16. Calder, M., y Sevagnani, M. (2014). Modelling ieee 802.11 csma/ca rts/cts with stochastic bigraphs with sharing. *Formal Aspects of Computing*, 26(3), 537–561.
17. Cao, J., y Das, S. K. (2012). *Mobile agents in networking and distributed computing* (Vol. 3). John Wiley & Sons.
18. Chandra, T. D., y Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2), 225–267.
19. Chelladhurai, J. S., Singh, V., y Raj, P. (2017). *Learning docker*. Packt Publishing Ltd.
20. Committee, I. . L. S., y cols. (2012). Ieee standard for information technology-telecommunication and information exchange between systems-local and metropolitan area networks-specific requirements part11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendmentl: Radio resource measurement of wireless lans. <http://standards.ieee.org/getieee802/download/802.11n-2012>.
21. Correa, B. A., Ospina, L., y Hicapie, R. C. (s.f.). Técnicas de agrupamiento para redes móviles ad hoc. *Rev. fac. ing. univ. Antioquia*, 145–161.
22. de Sousa, N. F. S., Perez, D. A. L., Rosa, R. V., Santos, M. A., y Rothenberg, C. E. (2018). Network service orchestration: A survey. *arXiv preprint arXiv:1803.06596*.
23. Dolui, K., y Datta, S. K. (2017). Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. En *Global internet of things summit (giots), 2017* (pp. 1–6).
24. Fan, B., Tian, H., Zhang, Y., y Yan, X. (2016). Resource allocation in a generalized framework for virtualized heterogeneous wireless network. *Mobile Information Systems*, 2016.
25. Feeney, L. M., y Nilsson, M. (2001). Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. En *Infocom 2001. twentieth annual joint conference of the ieee computer and communications societies. proceedings. ieee* (Vol. 3, pp. 1548–1557).

26. Ferscha, A., Farrahi, K., Van den Hoven, J., Hales, D., Nowak, A., Lukowicz, P., y Helbing, D. (2012). Socio-inspired ict: Towards a socially grounded society-ict symbiosis. *The European Physical Journal Special Topics*, 214 (1), 2012.
27. Fischer, M. J., Lynch, N. A., y Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2), 374–382.
28. Fitoussi, D., y Tennenholz, M. (2000). Choosing social laws for multi-agent systems: Minimality and simplicity. *Artificial Intelligence*, 119(1-2), 61–101.
29. Fitzek, F. H., y Katz, M. D. (2013). *Mobile clouds: Exploiting distributed resources in wireless, mobile and social networks*. John Wiley & Sons.
30. Flasiński, M. (2016). *Introduction to artificial intelligence*. Springer.
31. Gao, M., Addis, B., Bouet, M., y Secci, S. (2017). Optimal orchestration of virtual network functions. *arXiv preprint arXiv:1706.04762*.
32. Gavalas, D., Pantziou, G., Konstantopoulos, C., y Mamalis, B. (2006). Clustering of mobile ad hoc networks: an adaptive broadcast period approach. En *Communications, 2006. icc'06. ieee international conference on* (Vol. 9, pp. 4034–4039).
33. Gilbert, S., y Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, 33(2), 51–59.
34. Gilbert, S., y Lynch, N. (2012). Perspectives on the cap theorem. *Computer*, 45(2), 30–36.
35. Giotis, K., Kryftis, Y., y Maglaris, V. (2015). Policy-based orchestration of nfv services in software-defined networks. En *Proceedings of the 2015 1st ieee conference on network softwarization (netsoft)* (p. 1-5). doi: 10.1109/NETSOFT.2015.7116145
36. Goldberg, R. P. (1973). Architecture of virtual machines. En *Proceedings of the workshop on virtual computer systems* (pp. 74–112).
37. Gu, L., Zeng, D., Guo, S., Barnawi, A., y Xiang, Y. (2017). Cost efficient resource management in fog computing supported medical cyber-physical system. *IEEE Transactions on Emerging Topics in Computing*, 5(1), 108–119.
38. Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., y Brown, R. B. (2001). Mibench: A free, commercially representative embedded benchmark suite. En *Workload characterization, 2001. wwc-4. 2001 ieee international workshop on* (pp. 3–14).
39. Haddad, S., Kordon, F., Pautet, L., y Petrucci, L. (2013). *Distributed systems: Design and algorithms*. John Wiley & Sons.

40. Hayashibara, N., Cherif, A., y Katayama, T. (2002). Failure detectors for large-scale distributed systems. En *Reliable distributed systems, 2002. proceedings. 21st ieee symposium on* (pp. 404–409).
41. Henz, M., Smolka, G., y Würtz, J. (1993). Oz-a programming language for multi-agent systems. En *Ijcai* (pp. 404–409).
42. Hobbes, T. (2001). *Leviatán, madrid.* Alianza.
43. Hu, P., Dhelim, S., Ning, H., y Qiu, T. (2017). Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications*.
44. Huang, Y., y Li, G. (2010). A semantic analysis for internet of things. En *Intelligent computation technology and automation (icicta), 2010 international conference on* (Vol. 1, pp. 336–339).
45. Ilyas, M. (2002). *The handbook of ad hoc wireless networks.* CRC press.
46. Jepsen, T. C. (2003). *Distributed storage networks: architecture, protocols and management.* John Wiley & Sons.
47. Jung, J. L., Jaques, P. A., de Andrade, A. F., y Vicari, R. M. (2002). The conception of agents as part of a social model of distance learning. En *Brazilian symposium on artificial intelligence* (pp. 140–151).
48. Kang, S., Lee, Y., Min, C., Ju, Y., Park, T., Lee, J., ... Song, J. (2010, March). Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments. En *2010 ieee international conference on pervasive computing and communications (percom)* (p. 135-144). doi: 10.1109/PERCOM.2010.5466982
49. Khaitan, S. K., y McCalley, J. D. (2015). Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal*, 9(2), 350–365.
50. Kuhn, T. S. (2012). *The structure of scientific revolutions.* University of Chicago press.
51. Kumar, M., Kulkarni, A. J., y Satapathy, S. C. (2018). Socio evolution & learning optimization algorithm: A socio-inspired optimization methodology. *Future Generation Computer Systems*, 81, 252–272.
52. Lamport, L., Shostak, R., y Pease, M. (1982). The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3), 382–401.
53. Lee, J., Bagheri, B., y Kao, H.-A. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3, 18–23.
54. Li, Q., y Negi, R. (2011). Distributed scheduling in cyber-physical systems: The case of coordinated electric vehicle charging. En *Globecom workshops (gc wkshps), 2011 ieee* (pp. 1183–1187).

55. Liang, C., y Yu, F. R. (2015). Wireless network virtualization: A survey, some research issues and challenges. *Communications Surveys & Tutorials, IEEE*, 17(1), 358–380.
56. Licklider, J. C. (1960). Man-computer symbiosis. *IRE transactions on human factors in electronics*(1), 4–11.
57. Loo, J., Mauri, J. L., y Ortiz, J. H. (2016). *Mobile ad hoc networks: current status and future trends*. CRC Press.
58. Lynch, N. A. (1996). *Distributed algorithms*. Elsevier.
59. Mahmud, R., Kotagiri, R., y Buyya, R. (2018). Fog computing: A taxonomy, survey and future directions. En *Internet of everything* (pp. 103–130). Springer.
60. Mansutti, A., Miculan, M., y Peressotti, M. (2014). Multi-agent systems design and prototyping with bigraphical reactive systems. En *Ifip international conference on distributed applications and interoperable systems* (pp. 201–208).
61. Marca, D. A., y McGowan, C. L. (1987). *Sadt: structured analysis and design technique*. McGraw-Hill, Inc.
62. Matthias, K., y Kane, S. P. (2015). *Docker: Up & running: Shipping reliable containers in production*. O'Reilly Media, Inc.
63. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2), 69–74.
64. Miell, I., y Sayers, A. H. (2016). *Docker in practice*. Manning Publications Co.
65. Milner, R. (1999). *Communicating and mobile systems: the pi calculus*. Cambridge university press.
66. Milner, R. (2008). Bigraphs and their algebra. *Electronic Notes in Theoretical Computer Science*, 209, 5–19.
67. Milner, R. (2009). *The space and motion of communicating agents*. Cambridge University Press.
68. Mood, A. M. (1950). *Introduction to the theory of statistics*.
69. Moore, P., y Roger, K. (2016). Pct and beyond: Towards a computational framework for intelligent'communicative systems. *arXiv preprint arXiv:1611.05379*.
70. Mouat, A. (2015). *Using docker: Developing and deploying software with containers*. O'Reilly Media Inc.

71. Mouradian, C., Saha, T., Sahoo, J., Abu-Lebdeh, M., Glitho, R., Morrow, M., y Polakos, P. (2016). Network functions virtualization architecture for gateways for virtualized wireless sensor and actuator networks. *arXiv preprint arXiv:1601.03390*.
72. Nan, Y., Li, W., Bao, W., Delicato, F. C., Pires, P. F., y Zomaya, A. Y. (2018). A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems. *Journal of Parallel and Distributed Computing*, 112, 53–66.
73. Negash, B., Rahmani, A. M., Liljeberg, P., y Jantsch, A. (2018). Fog computing fundamentals in the internet-of-things. En *Fog computing in the internet of things* (pp. 3–13). Springer.
74. Neme, A., y Hernández, S. (2009). Algorithms inspired in social phenomena. En *Nature-inspired algorithms for optimisation* (pp. 369–387). Springer.
75. Neumann, A., Aichele, C., Lindner, M., y Wunderlich, S. (2008). Better approach to mobile ad-hoc networking (batman). *IETF draft*, 1–24.
76. Newman, S. (2015). *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc.
77. Nitschke, G. (2005). Emergence of cooperation: State of the art. *Artificial Life*, 11(3), 367–396.
78. Ongaro, D., y Ousterhout, J. K. (2014). In search of an understandable consensus algorithm. En *Usenix annual technical conference* (pp. 305–319).
79. Perrin, C. (2008). The cia triad. *Dostopno na: http://www.techrepublic.com/blog/security/the-cia-triad/488*.
80. Peterson, L., y Roscoe, T. (2006). The design principles of planetlab. *ACM SIGOPS operating systems review*, 40(1), 11–16.
81. Pitt, J., Busquets, D., y Riveret, R. (2015). The pursuit of computational justice in open systems. *AI & society*, 30(3), 359–378.
82. Portnoy, M. (2012). *Virtualization essentials* (Vol. 19). John Wiley & Sons.
83. Ramanathan, R., Allan, R., Basu, P., Feinberg, J., Jakllari, G., Kawadia, V., ... Freebersyser, J. (2010). Scalability of mobile ad hoc networks: Theory vs practice. En *Military communications conference, 2010-milcom 2010* (pp. 493–498).
84. Rawls, J. (2012). *Teoría de la justicia*. Fondo de cultura económica.
85. Raychaudhuri, D., y Mandayam, N. B. (2012, abril). Frontiers of Wireless and Mobile Communications. *Proceedings of the IEEE*, 100(4), 824–840. Descargado de <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6155060> doi: 10.1109/JPROC.2011.2182095

86. Sangiorgi, D., y Walker, D. (2003). *The pi-calculus: a theory of mobile processes*. Cambridge university press.
87. Santoro, D., Zozin, D., Pizzolli, D., De Pellegrini, F., y Cretti, S. (2017). Foggy: a platform for workload orchestration in a fog computing environment. En *Cloud computing technology and science (cloudcom), 2017 ieee international conference on* (pp. 231–234).
88. Santos, J., Wauters, T., Volckaert, B., y De Turck, F. (2017). Fog computing: Enabling the management and orchestration of smart city applications in 5g networks. *Entropy*, 20(1), 4.
89. Sarkar, S. K., Basavaraju, T., y Puttamadappa, C. (2007). *Ad hoc mobile wireless networks: principles, protocols and applications*. CRC Press.
90. Sartre, J.-P. (1946). *El ser y la nada*. Iberoamericana.
91. Shahzadi, S., Iqbal, M., Dagiuklas, T., y Qayyum, Z. U. (2017). Multi-access edge computing: open issues, challenges and future perspectives. *Journal of Cloud Computing*, 6(1), 30.
92. Silberschatz, A., Galvin, P. B., y Gagne, G. (2014). *Operating system concepts essentials*. John Wiley & Sons, Inc.
93. Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., y Leitner, P. (2017). Optimized IoT service placement in the fog. *Service Oriented Computing and Applications*, 11(4), 427–443. doi: 10.1007/s11761-017-0219-8
94. Sood, S. K. (2018). Sna based qos and reliability in fog and cloud framework. *World Wide Web*, 1–16.
95. Spinoza, B. (s.f.). De,[1677] 21977. *Etica demostrada según el orden geométrico*.
96. Srinivasan, S., y Ramakrishnan, S. (2012). Cultural algorithm toolkit for multi-objective rule mining. *arXiv preprint arXiv:1209.2948*.
97. Sun, C., Bi, J., Zheng, Z., y Hu, H. (2017). Hyper: A hybrid high-performance framework for network function virtualization. *IEEE Journal on Selected Areas in Communications*, 35(11), 2490–2500.
98. Tiesel, P. S., Enghardt, T., Palmer, M., y Feldmann, A. (2018). Socket intents: Os support for using multiple access networks and its benefits for web browsing. *arXiv preprint arXiv:1804.08484*.
99. Tomic, S., Pecora, F., y Saffiotti, A. (2018). Norms, institutions, and robots. *arXiv preprint arXiv:1807.11456*.
100. Touati, Y., Daachi, B., y Arab, A. C. (2017). *Energy management in wireless sensor networks*. Elsevier.
101. Turing, A. M. (2009). Computing machinery and intelligence. En *Parsing the turing test* (pp. 23–65). Springer.

102. van de Belt, J., Ahmadi, H., y Doyle, L. E. (2017). Defining and surveying wireless link and network virtualization. *arXiv preprint arXiv:1705.03768*.
103. Van Roy, P. (2005). Multiparadigm programming in mozart/oz. En *Second international conference: revised, selected, and invited papers. lecture notes in computer science* (Vol. 3389).
104. Van-Roy, P., y Haridi, S. (2004). *Concepts, techniques, and models of computer programming*. MIT press.
105. Varela, C. A., y Agha, G. (2013). *Programming distributed computing systems: A foundational approach*. MIT Press.
106. Vico, G., De La Villa, R., Beccaría, J. M. R., Pompa, L., y Sepúlveda, S. D. (1995). *Ciencia nueva*. Tecnos Madrid.
107. Vinayakray-Jani, P., y Sanyal, S. (2012). Security architecture for cluster based ad hoc networks. *arXiv preprint arXiv:1207.1701*.
108. Watt, S. N. (1997). Artificial societies and psychological agents. En *Software agents and soft computing towards enhancing machine intelligence* (pp. 27–41). Springer.
109. Wen, H., Tiwary, P. K., y Le-Ngoc, T. (2013). *Wireless virtualization*. Springer.
110. Wittgenstein, L. (2009). *Philosophical investigations*. John Wiley & Sons.
111. Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.
112. Xiao, Y., y Pan, Y. (2009). *Emerging wireless lans, wireless pans, and wireless mans: ieee 802.11, ieee 802.15, 802.16 wireless standard family* (Vol. 57). John Wiley & Sons.
113. Yu, J. Y., y Chong, P. H. (2005). A survey of clustering schemes for mobile ad hoc networks. *IEEE Communications Surveys & Tutorials*, 7(1), 32–48.
114. Zaalouk, A., Khondoker, R., Marx, R., y Bayarou, K. (2014, May). Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions. En *2014 ieee network operations and management symposium (noms)* (p. 1-9).
115. Zhang, N., Yang, P., Zhang, S., Chen, D., Zhuang, W., Liang, B., y Shen, X. S. (2017). Software defined networking enabled wireless network virtualization: Challenges and solutions. *IEEE Network*, 31(5), 42–49.
116. Zhao, S., y Jain, S. (s.f.). Ad hoc and mesh network protocols and their integration with the internet. *Emerging Wireless Technologies and the Future Mobile Internet*, 54.
117. Zhou, Q., Wang, C.-X., McLaughlin, S., y Zhou, X. (2015). Network virtualization and resource description in software-defined wireless networks. *Communications Magazine, IEEE*, 53(11), 110–117.