



TAMPERE UNIVERSITY OF TECHNOLOGY

NADIR JAVED
IOT NODE EMULATION AND MANAGEMENT TESTBED

Master of Science thesis

Examiners: Prof. Jarmo Harju
MSc. Bilhanan Silverajan
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 8th May 2013

ABSTRACT

NADIR JAVED: IoT Node Emulation and Management Testbed
Tampere University of Technology
Master of Science thesis, 53 pages
December 2014
Master's Degree Programme in Information Technology
Major: Communications Engineering
Examiners: Prof. Jarmo Harju, MSc. Bilhanan Silverajan
Keywords: IoT, PlanetLab, Emulation, Dummynet, Internet of Things

There has been an explosive growth in the number of devices connected to the Internet during the past few years. These connected devices have already outnumbered the world's population and are expected to grow at the same or even higher rate in the future. There are some studies showing estimates of as high as 50 billion connected devices by the year 2020. This phenomenon of connecting physical objects to the Internet where they can intercommunicate as well as provide contextually aware data to the users is termed as Internet of Things.

Internet of Things although providing many useful applications and services to the users brings with itself a few challenges as well. The novelty presented by the Internet of Things is not the functionalities or the services provided by the smart objects but the sheer number of these smart objects. This gigantic amount of devices and the traffic they generate presents a lot of challenges especially related to the management of these objects and as well as how to adopt and utilize the generated data efficiently.

In order to address these problems and to provide a platform through which the users can easily recreate such IoT scenarios, investigate communication between devices and networks, execute services and monitor application level behavior, a solution was developed as part of this thesis. The developed platform works on top of an internationally distributed testbed called PlanetLab .

The developed system allows the users to define and emulate IoT devices and network interfaces. The interfaces can be defined based on bandwidth availability, time delay and packet loss ratio experienced over the interface. The system automatically generates the appropriate commands for emulating the configured devices and reverts back to the users with details on how to utilize the emulated device. It also provides support for associating tags with the available nodes and devices, which can be used to perform lookups and form organized viewpoints in the system.

PREFACE

This MSc thesis was completed at the Department of Pervasive Computing, Tampere University of Technology and the work was supported by the Internet of Things research program from Finnish strategic center for science, technology and innovation in the field of ICT.

I would like to start by thanking my supervisors, Bilhanan Silverajan for introducing me to the topic and guiding me through out the process of compiling this work and Jarmo Harju for providing me the opportunity to work on this thesis.

I would also like to thank my loving girlfriend Yulia, without her support and belief this work would have never been completed.

Finally I would like to thank my parents and my sister, who have always stood by me and their unfaltering belief in me has been the source of motivation for me throughout my life.

Helsinki, 12.11.2014

Nadir Javed

TABLE OF CONTENTS

1. Introduction	1
2. Background	3
2.1 Introduction to IoT	3
2.2 What is a Testbed?	4
2.3 Introduction to PlanetLab	5
2.4 PlanetLab Usage Scenarios	8
2.5 Terminology and Concepts Used in PlanetLab	9
2.6 Getting Started with PlanetLab	11
2.6.1 Registering with PlanetLab	11
2.6.2 Creating and Uploading the SSH Keys	12
2.6.3 Creating the Slices	13
2.6.4 Adding Nodes to the Slice	14
3. System Architecture and Design	16
3.1 System Concept	16
3.2 System Architecture	16
4. Implementation	19
4.1 Programming Languages and Technologies Used	19
4.2 Detailed System Design	20
4.3 System Application Flow	26
4.4 System Usage and User Interfaces	30
4.4.1 Defining Network Interfaces	30
4.4.2 Defining Devices & Associating Interfaces	32
4.4.3 Adding and Managing the Physical Nodes	34
4.4.4 Emulating and Managing the Devices	36
4.4.5 Adding Tagging Information to Devices and Hosts	40
5. Results and Future Work	45
5.1 Verification and Results	45

5.2 Future Work	48
6. Conclusion	50
References	51

LIST OF FIGURES

2.1	Global distribution of PlanetLab nodes.	6
2.2	Registration process for PlanetLab access.	12
2.3	Adding nodes to the slice.	14
3.1	System Concept	17
3.2	System Architecture	18
4.1	Detailed Implementation Design	20
4.2	Communication between User and Management Server	22
4.3	Back-end Communication between Management Server, PlanetLab Central Server and Remote Host	23
4.4	System logic for device emulation: Message sequence chart part 1 . .	27
4.5	System logic for device emulation: Message sequence chart part 2 . .	28
4.6	System logic for device emulation: Message sequence chart part 3 . .	29
4.7	User Interface: Defining Network Interfaces	30
4.8	Database Table for Storing Network Interfaces	31
4.9	User Interface: A list of all available network interfaces	32
4.10	User Interface: Defining Devices and Associating Network Interfaces .	33
4.11	Database Tables for Storing Device Information	33
4.12	User Interface: List of all available devices and their details	34
4.13	User Interface: Adding physical nodes to the system.	35
4.14	Database Tables for Storing Node Information	36
4.15	User Interface: List of all available nodes and their details	37

4.16 User Interface: Emulating devices on PlanetLab nodes.	37
4.17 Database Tables for Storing Information about Emulated Devices. . .	38
4.18 User Interface: A list of all emulated devices in the system.	39
4.19 User Interface: A detailed summary of an emulated devices.	40
4.20 User Interface: Adding new tags & listing of all available tags and their descriptions.	41
4.21 User Interface: Associating tags to devices and nodes.	42
4.22 Database tables for storing information about tags.	43
4.23 User Interface: Search using associated tags.	43
5.1 Outgoing traffic bitrate verification.	46
5.2 Incoming traffic bitrate verification.	47
5.3 Time delay affecting bitrate verification.	47
5.4 Packet loss ratio affecting bandwidth measurements.	48

LIST OF TABLES

4.1 List of roles available for network emulation using netconfig	25
-----------------------------------------------------------------------------	----

ABBREVIATIONS AND TERMS

AJAX	Asynchronous JavaScript
API	Application Programming Interface
BLE	Bluetooth Low Energy
CIFS	Common Internet File System
DBMS	Database Management System
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
IoT	Internet of Things
IP	Internet Protocol
IPFW	Internet Protocol Firewall
LAN	Local Area Network
LTE	Long Term Evolution
OS	Operating System
P2P	Peer to Peer
PC	Personal Computer
PHP	Hypertext Preprocessor (Programming language)
QoS	Quality of Service
RFID	Radio-frequency Identification
RPC	Remote Procedure Call
SQL	Structured Query Language
SSH	Secure Shell
UI	User Interface
URL	Uniform Resource Locator
VM	Virtual Machine
VMM	Virtual Machine Monitor
WLAN	Wireless Local Area Network also referred to as Wi-Fi
XML	Extensible Markup Language

1. INTRODUCTION

The number of Internet connected devices had already outnumbered the world's human population in 2012 exceeding the figure of 8.7 billion connected devices [36]. This number has continued to grow over the years and already by 2014 has exceeded 16 billion devices [34]. With this number already being forecasted to be much higher in the next coming year, studies are showing that by the year 2020 the number of connected devices would significantly overshadow the current number and would reach a figure ranging from 40 to 50 billion connected devices [25][24].

These connected devices are composed of both smart devices as well as the resource constrained devices such as sensors and actuators. These devices in addition to common network connectivity interfaces such as Wi-Fi, Cellular connection and Ethernet also possess a variety of low power wireless communication technologies such as BLE (Bluetooth Low Energy) and Zigbee.

This large diversity of connected nodes and their connectivity interfaces impacts the types of service interactions and network communications, in both client-server as well as peer to peer configurations. Smart devices such as phones and tablets nowadays possess hardware that allows multiple radio technologies to co-exist, which gives rise to the possibility of multipath communication. Gateway nodes can enable network traffic generated by a device from one kind of communication technology to easily access and traverse into another kind.

Wireless sensors as per se might sometimes not possess direct connectivity to the Internet but they can reach the Internet usually by utilizing multi-hop relays where one of the end nodes acts as a gateway and forwards the traffic to the Internet. Apart from the kind of radio technology used, there is also great variation in bandwidth characteristics and latency experienced over these links. The reliability of these links is also variable and might sometimes vary even among the same family of links.

The existence of these diverse nodes and interfaces plays an important role in forming the current architecture of the Internet and gives rise to many challenges when studying or experimenting with network communications over the current or future

Internet architecture. Problems such as measuring traffic flows among disparate types of networks, and studying traffic characteristics of pairwise node-based interactions are not trivial. The management of these devices and nodes, as well as performing lookups and discovering devices are challenging problems to counter especially given the scale of deployment of these nodes and networks. Services and application-level behavior also are influenced by these devices and network architectures and need to adapt accordingly.

In order to address such problems and allow investigation to occur, from device to the network, or to subsequently execute services and monitor application level behavior, a platform was created as part of this work, which provided scalable distributed node emulation architecture and possibility to emulate network interactions based on device-level interface characteristics and network conditions. This platform was developed atop an international distributed testbed called PlanetLab.

While creating the platform, the primary aim of the work was to provide a platform for testing that allows the emulation of devices providing insights into the use and behavior of smart devices as well as resource constrained nodes in the Internet of Things. The work aimed to provide network heterogeneity, flexibility, scalability and allow remote node management.

The rest of the thesis is structured as follows. Chapter 2 provides an introduction to testbeds and Internet of Things and discusses some background work related to PlanetLab. Chapter 3 presents the architecture and design of the developed platform and Chapter 4 provides the implementation details of the developed system. Chapter 5 discusses the measurement results taken for verification and the future work that can be carried out. Chapter 6 concludes the thesis with a brief summary of the work.

2. BACKGROUND

This chapter caters towards providing a brief overview of the platforms and concepts that are most relevant to the work carried out as part of this thesis. It gives an introduction to the concepts of IoT and Testbeds. PlanetLab system on top of which the work was carried out is also introduced along with the common usage scenarios. A small overview on how to use the PlanetLab system is also provided as part of this chapter.

2.1 Introduction to IoT

The Internet of Things or IoT when put simply is the connection of physical things to Internet, which makes it possible to share data and control physical objects or access services from a distance. The IoT concept builds up on smaller blocks commonly referred to as smart devices or smart objects. These are basically just embedded devices that possess connectivity to the Internet. The concept of IoT has been around for a long while, first popularized with the introduction of RFID (Radio-frequency Identification). It introduced some sort of intelligence to simple identification tags found on everyday objects allowing their identification to be decoded from a distance. By introducing more intelligence to the ID tags, the tagged items become smart items or objects. The concept of Internet of Things is not in introduction of a disruptive technology, but in the pervasive deployment of smart objects [30].

The novelties present in the world of IoT are not related to the functionality these smart objects offer or even the communication technologies available on these devices. The real novelty that the IoT presents is the share deployment scale of these devices, the number of connected devices keeps increasing with each year and already has exceeded the number of human population on the world. Due to these large-scale deployments, new challenges and problems arise in dealing with management of these nodes and maintaining the connectivity among these devices.

The main problems that are faced in the field of IoT are authenticity and identification of the smart objects, automated management and self organizing networks

without any fixed topology. Diagnostic and maintenance required by the smart devices after deployment present new challenges as well as context awareness among the devices and application-level behavior of services utilizing the smart objects.

The nature of IoT also raises a lot of privacy concerns as gigantic amounts of data is generated by these smart devices which can range from just weather or traffic conditions to very personal details about an individual's behaviors and habits. The data needs to be anonymized or encrypted, depending on the kind of service that is going to be utilizing this data and as well as fine grained control needs to be provided to the users owning these smart object so they can control exactly when and how a service or application can access this data.

2.2 What is a Testbed?

Testbed can be defined as a platform designed for carrying out testing and experimentation in a specific field or for a specific purpose. Most of the testbeds focus on one or more very closely related scientific areas and provide a general architecture which would be suitable for the applications or services in that particular domain for which the testbed was designed for. Testbeds focus on providing resources to the testers and researchers, which are reusable in a sense they can easily be configured or changed for other experiments and would be too cumbersome, time consuming or expensive to be setup for individual projects and experiments.

Testbeds in the world of computer science focus on providing computing resources to the users, which can be easily managed from a single control point and adjusted to change the topology in which it is organized or increase and decrease computing power depending on the nature of experiment. In light of the work carried out for this thesis project, the testbeds providing features and functionalities for research in the field of network communication were of particular interest and some of the most related testbeds are introduced briefly.

The MagNets project [29] is aimed at deploying a next generation wireless access network testbed infrastructure. The testbed is based in city of Berlin and through this testbed, heterogeneous devices possessed by the students in the university are allowed free access to an operator supported network.

The Pan-European Laboratory testbed [38] provides a resource federation framework, which allows multi-domain testbeds. These testbeds provide a heterogeneous cross layer infrastructure for broad testing and experimentation. The SmartSantander [35] project deals with providing a city-wide test facility for experimentation

of architectures and enabling services and applications for the IoT. It is a platform that provides for large-scale experimentation in real life conditions.

There are some testbeds available as well that target specific radio communication technologies and once such example is a unified testbed platform developed to emulate LTE over wired Ethernet [23]. It can be used to examine the key aspects of a LTE System in real-time, including real-time uplink and downlink scheduling, quality of service parameters and Android [1] end-user applications. The Distributed Network Emulator (DNEmu) [37] provides facilities for investigating how realistic network experiment can be performed involving globally distributed physical nodes under heterogeneous environments.

A very relative testbed available through OneLab [26] provides a platform with physical wireless and mobile devices. This testbed is especially targeted towards Internet of Things and offers both fixed and mobile nodes. The mobile nodes can be controlled through robots and model trains. The environment these nodes are present in can also be controlled and ranges from rooms wrapped in faraday cages to isolate away from normal radio interference to normal office building environment.

The most relevant testbed for the work carried out in this thesis was found to be PlanetLab [11] as it offered a versatile choice of physical hosts around the globe and the sheer number of nodes available made it an automatic choice for this work. As the work carried out for this thesis demanded a platform that could handle emulated nodes from tens of hundred to tens of thousand, PlanetLab appeared to be the most suitable choice. PlanetLab is described in more detail in the section 2.3.

2.3 Introduction to PlanetLab

PlanetLab is a geographically distributed collection of computers that serves as a testbed for aiding research and development activities especially in the field of computer networking and distributed systems research. It is a platform for researching, developing and deploying planetary scale services and experiments over a highly distributed and heterogeneous environment [13].

It is a system that allows for easy deployment of services or a distributed applications over a large amount of physical nodes distributed globally. It provides means for easily monitoring the results and managing the connection between the active nodes. These nodes are all connected to the Internet and are managed centrally by the PlanetLab servers monitored by the PlanetLab administration body.

At the time of writing PlanetLab has about 1317 physical nodes spread around the

globe in approximately 630 sites [11]. There are more than 1000 different experiments that are being run over the overlay network that the PlanetLab provides for testing and researching activities. A spread of the PlanetLab nodes over a world map to highlight the geographical dispersion of the nodes is represented in Figure 2.1.

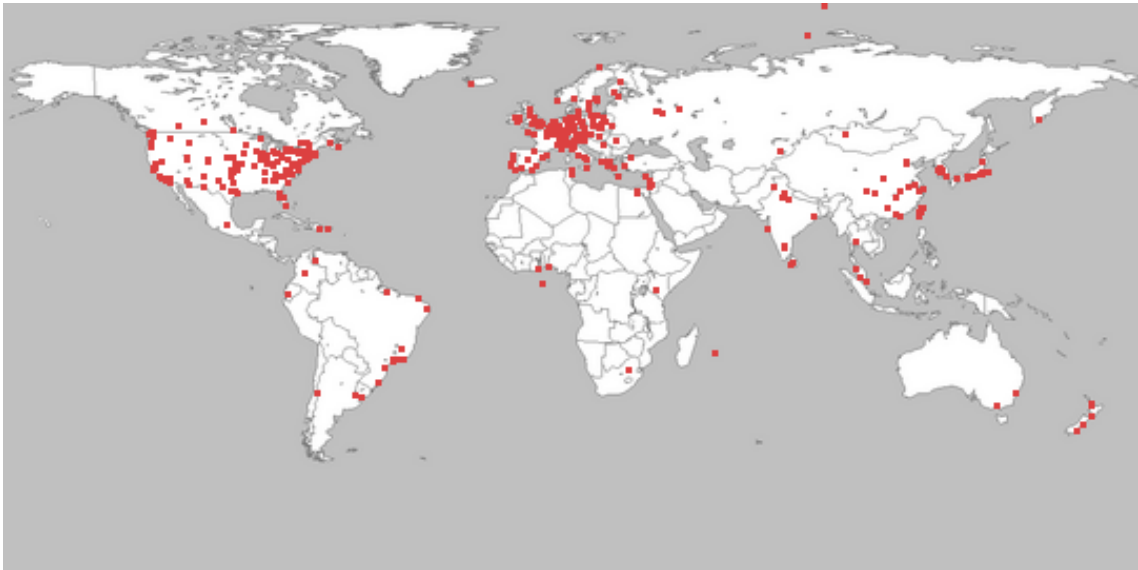


Figure 2.1 An illustration indicating the global dispersion of PlanetLab nodes [11].

The universities and research institutes around the world host most of the PlanetLab nodes and are managed by a community of researchers. By hosting a small number of nodes, an organization or research body can become part of the PlanetLab consortium and gain access to a share of resources across the network of PlanetLab nodes [31]. The size of the share depends upon the amount of nodes hosted by the organization and the membership level in the consortium.

All of the PlanetLab nodes run a custom version of Fedora [4], which is a Linux based operating system and the operating system running on the nodes itself is called MyPLC [7]. MyPLC operating system includes mechanisms for controlling system configurations on the node, remotely distributing software updates and a set of management tools. It also provides methods for monitoring the status of nodes and system activity to prevent illegal usage of the available resources. There are some mechanisms available as well for managing user accounts and distributing the keys associated with the accounts for allowing remote access to the nodes on which the software is installed.

MyPLC along with being available on the PlanetLab nodes is also distributed as a software package available to all members. This enables the users to even install it

on local nodes and have a private PlanetLab like testbed available for running their experiments. The advantage being, the users will have access to full set of resources available on each node without having to share with other ongoing experiments. The custom version of PlanetLab is especially beneficial for experiments that require special architecture on the node or an uninterrupted access to large amount of resources while still being able to leverage the distributed nature of PlanetLab architecture.

The main feature offered by the MyPLC software is the ability to dynamically create and destroy VMs (virtual machines) on the remote nodes and to achieve this in a distributed fashion. The distribution here refers to the individual virtual machines being deployed and controlled on all the nodes that are part of the project. MyPLC allows the creation and management of the virtual machines through a single control point and the ability to control the complete set of virtual machines as a single compounded entity. MyPLC employs the use of VMMs (Virtual Machine Monitor) for monitoring and controlling the VMs on each node.

Each virtual machine running on a node is allocated a share of that node's resources and a single node could have several virtual machines running simultaneously. This allows the users of PlanetLab to have simultaneous access to all the available nodes and execute their services and applications in complete isolation from each other. This isolation is not only based on how the physical resources of a node are accessed by the VMs but also on the network communication of the VMs which remain totally independent of other VMs[32] although the node has just one IP (Internet Protocol) address through which it can be reached.

The collection of virtual machines running on each node that is part of the user's project is treated as a single compounded entity in PlanetLab and is termed as a *Slice*. According to the design principles of PlanetLab, this collection of VMs is a global abstraction and neither the VMM nor the PlanetLab nodes and the node manager running on the nodes are aware of this global abstraction. These entities in turn just deal with creation and management of single virtual machines in order to keep these tasks as simple as possible [32]. The slice abstraction and management are handled by special slice creation services available only to the PlanetLab central servers thus simplifying the logic of creating slices and keeping it manageable by keeping the control plane at single point in the network.

2.4 PlanetLab Usage Scenarios

PlanetLab has been designed to address a wide range of problems faced in the field of research activities related to computer networks and communication over the existing Internet architecture. In this section a summary is presented on some sample use case scenarios for utilizing PlanetLab and also some of the scenarios albeit possible with PlanetLab but are not allowed or encouraged to be executed over the PlanetLab testbed.

PlanetLab presents an excellent facility and infrastructure for running and experimenting with widely distributed services. These service could be related to telecommunication networks, networked applications, real-time process control applications, distributed rendering services or parallel computing.

All the experiments dealing with any sort of communication can easily benefit from the globally dispersed resources available through PlanetLab. Studies dealing with wireless sensor networks, routing algorithms or even P2P (peer to peer) oriented application are the best suited to leverage the capabilities provided by PlanetLab.

Networked applications are also highly suited to run over PlanetLab, as the nature of these applications demands an architecture very close to the one provided by PlanetLab. Some examples of such networked application include distributed DBMS (database management systems), large scale multiplayer online games, networked file systems such as CIFS (Common Internet File System) [2] and globally distributed information processing systems such as banking and airline reservation systems.

Some other use case scenarios suited for experimentation over PlanetLab include applications that deal with real time process controlling such as Air traffic control systems or industrial control systems that are dependent on communication between a wide array of sensors. The biggest challenge faced by these applications is that the core functionality of the system should remain always intact and responsive even if some of the sensors are unresponsive or unreachable. This kind of scenario is easily reproducible over PlanetLab where one of the nodes could act as the central controller and the remaining nodes could be configured to take the role of the sensors. The availability of the nodes acting as sensors could be easily toggled using the controls exposed by the PlanetLab allowing researchers to easily reproduce and monitor the behavior of the applications in a controlled environment.

PlanetLab not only allows the users to easily reproduce different topologies and scenarios for monitoring the behavior of networked applications but also provides an environment for actual testing and deployment, which is very close to real Internet

behavior. As all the PlanetLab nodes are connected to the Internet and possess no means of directly communicating with each other without using the Internet, they are as unpredictable as any other connected node over the Internet. However, they cannot mimic the unpredictability of Internet nodes entirely as most of these nodes are hosted by the universities and research centers around the world which tend to have very reliable connectivity to the internet with an uplink and downlink capacity at the very high end as compared to an average connected node. The downtimes of these nodes are also very minimal when compared to an average connected node of Internet as the organizations hosting the PlanetLab nodes often have backup power generators and back up network connectivity links.

Although there are some applications that are well suited to be run over PlanetLab, they are not allowed or encouraged to utilize the resources offered by the PlanetLab platform. One of the first examples of such use case is to utilize PlanetLab as a distributed supercomputer. Although PlanetLab nodes collectively can offer a huge amount of computing resources when grouped together, the purpose of these node is not to offer raw computing power. The primary purpose of PlanetLab is to provide a platform for experimenting and deploying services and applications that require an architecture with a lot of nodes involved and present a big barrier to researchers dealing with such experiments.

PlanetLab is not a simulation platform and does not provide any simulation tools on its nodes. The nodes are actual physical machines and are connected to each other only by the means of Internet and do not provide any kind of simulated connection between them. PlanetLab should also not be used as a grid computer, which although a possibility, is not the purpose of the platform. However, research or experimentation related to studying the connectivity between grid computing nodes, trying out new methods and testing novel theories on how this communication is carried out would be a very suitable topic to utilize the resources offered by PlanetLab.

One last thing to keep in mind when utilizing PlanetLab is that its not an Internet emulator and as detailed before uses physical nodes and connections. Another thing to keep in mind is that although PlanetLab offers a distribution of heterogeneous nodes around the globe, its not a complete representative of the Internet and does not represent or encompass the complete architecture of current Internet.

2.5 Terminology and Concepts Used in PlanetLab

In this section some of the common terms used when dealing with PlanetLab testbed are introduced. These terms appear frequently when using PlanetLab or reading the

documentation and guidelines on utilizing PlanetLab. Its important to understand these terms and the underlying concepts before starting out with PlanetLab.

Principal Investigator

Principal investigator or more commonly referred to as *PI* in PlanetLab ecosystem are the users that are mainly responsible for managing resources, nodes and users at each site hosting a PlanetLab node. There always needs to be a user with the role of principal investigator at each PlanetLab site with some larger sites having the exception of more than one principal investigator. Usually the principal investigator is someone with an authoritative role within the organization hosting the PlanetLab node such as a faculty member at an educational institute or a project manager at a commercial organization.

User

Any one who utilizes PlanetLab for any kind of activity be it developing applications for PlanetLab or deploying services or conducting experiments over PlanetLab is considered a *User*. Principal Investigators could themselves be users as well. Every user needs to register with PlanetLab central and the membership needs to be approved by the principal investigator of the site to which the user belongs.

Site

Site is the term used in PlanetLab for referring to any physical location that is hosting one or more of the PlanetLab nodes. For example as Tampere University of Technology is a participating entity in PlanetLab and hosts a couple of nodes for PlanetLab, it is termed as a site in the PlanetLab terminology.

Node

Any physical computer that is a dedicated server running components of the PlanetLab service is termed a *Node* in PlanetLab. Every computer that is part of the PlanetLab network and is registered to the PlanetLab central servers as a participating machine is considered a Node in PlanetLab.

Slice

A global abstraction that combines the set of allocated resources across all the PlanetLab nodes assigned to a project is termed as *Slice*. As explained earlier in section 2.3 each node assigned to the project creates a virtual machine for that particular project exclusively and this collection of virtual machines can be controlled by the PlanetLab central servers as a single entity. This entity is called a Slice. These slices have a finite lifetime to conserve the wasting of resources if the project is no longer being actively worked upon but the slices can be renewed periodically to keep them active for as long as required across all the involved nodes.

Sliver

An instance of the slice running on a specific node is termed as *Sliver*. It is the smallest level of isolation in PlanetLab for the project or to term it in another way, a virtual machine running on a specific node assigned to a slice individually would be referred to as a sliver. It is actually the part of node's resources that is accessible to the users for their projects and can be accessed remotely by the users using SSH.

2.6 Getting Started with PlanetLab

There are a number of steps, which the users need to perform before they can start utilizing the resources offered by PlanetLab. These steps range from understanding the registration process to creating the project and the slices and finally how to access the system for executing the experiments. In this section a summary of these step is provided to simplify the process of getting started with PlanetLab.

2.6.1 Registering with PlanetLab

The first step the users need to perform before utilizing PlanetLab is to register with the PlanetLab central system. There are three PlanetLab central authorities to which the users can register with depending upon which central authority the user's home site belongs to. The three central authorities of PlanetLab are *PL-USA* (PlanetLab USA) [11], *PL-EU* (PlanetLab Europe) [13] and *PL-JP* (PlanetLab Japan).

PlanetLab central authorities all have online portals through which the services offered by each body can be accessed. These central services manage all the resources

and nodes registered to the body as well as all the users registered with that PlanetLab central service. Anyone requiring access to the PlanetLab system must first register with the appropriate PlanetLab website. After the registration users receive an email for verification and once its completed the principal investigator receives the request for approval of the new users. If granted by the principal investigator of the site, the user's account is created and they receive an confirmation email with details about the next steps required in the process. This registration process is presented as a flowchart diagram in Figure 2.2.

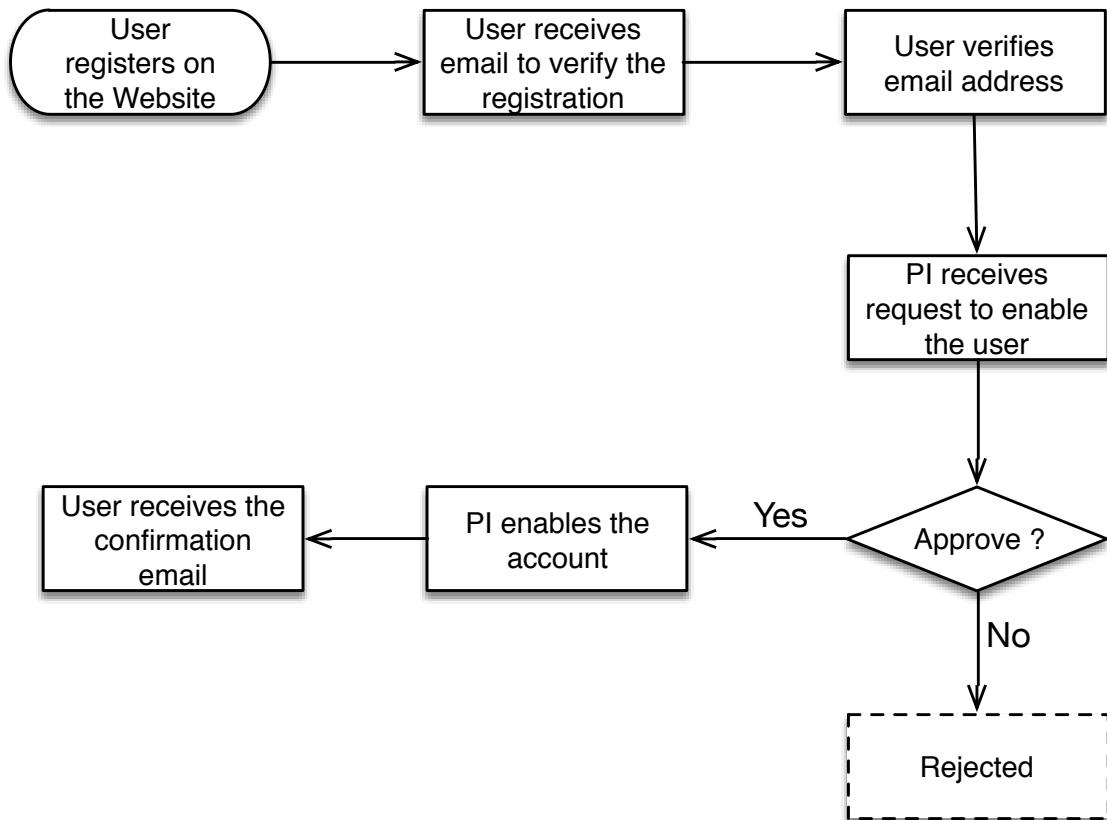


Figure 2.2 A flowchart depicting the registration process for an user to gain access to the PlanetLab system.

2.6.2 Creating and Uploading the SSH Keys

Access to the resources available through PlanetLab is secured by the use of public key encryption mechanism. Therefore the next step in the process is for the users to create a public and private key pair and upload it to the PlanetLab central website.

PlanetLab central's members area exposes an interface through which the public keys of the users can be uploaded to the system and then the key management

services of PlanetLab helps propagate the keys to all the nodes that are part of the user's slice. PlanetLab uses 1024 bit RSA keys for the authentication process on the nodes and its required for the users to generate the keys of at least 1024 bit. Although not a restriction it is strongly recommended to use keys with the added security of passphrases. Key pairs can easily be generated on Unix [18] based systems by issuing the following command through the command line interface or on widows systems by using programs such as PuttyGen [14].

```
[user@host#] sshkeygen -t rsa f /.ssh/id_planetlab
```

2.6.3 Creating the Slices

Once the users have created their account and uploaded their public keys to the central server, the next step in the process is to create a slice in PlanetLab that they would use for conducting their experiments. Only the users with principal investigator role are allowed to create new slices and the number of slices allowed per organization depends on the membership level of the organization in PlanetLab. Therefore the users need to contact the principal investigator of their site if they require a new slice to be created.

When creating a PlanetLab slice some information needs to be provided to the administration body, which describes the nature of experiments that would be executed on the slice. Some estimated processing power required by the slice and the network load the slice would generate also needs to be provided at the time of creation. A summary of the outcome or the expected results that the experiments running over the slice would achieve should also be provided.

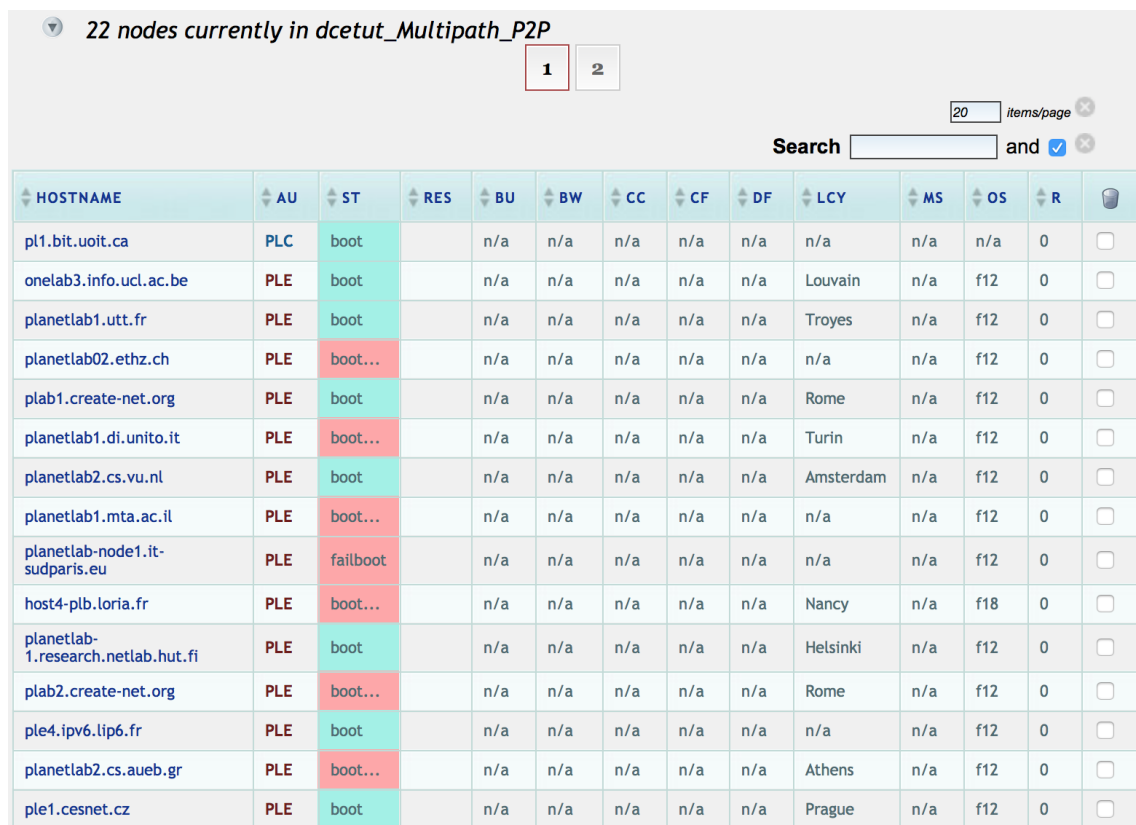
When the slice is ready on the PlanetLab central servers, the PI who created the slice is informed and then the PI could add users to the slice that belong to his site. The users receive an invitation to join the slice and upon joining they can start accessing the slice through PlanetLab central interface.

The slices are automatically cancelled and removed from the PlanetLab servers after a fix period of time unless they are renewed periodically. Any user who is assigned to the slice can renew it through the web interface provided by PlanetLab central. The expiration date is visible in the management interface of the slice and all users that belong to the slice are sent an email reminder to avoid any accidental removal of active slices.

2.6.4 Adding Nodes to the Slice

Once the slice is created, the next step in the process is to add the desired physical nodes available to the user through PlanetLab. The choice of which nodes to add to the slice is solely up to the user and PlanetLab does not restrict the users any how on which nodes to add and the quantity of nodes that can be added to a slice. The users can select freely from the pool of nodes based on their requirements, for example a user might choose based on specific physical locations, architecture of the node or the processing power available on the node.

PlanetLab provides two methods to the users through which the nodes can be added to slice. First method is to use the slice management tools provided by PlanetLab central through a web based interface [11]. These interfaces are highly configurable and the users can choose the kind of information they want to see for each node through this interface. The information is presented to the user in a tabular format and the users can then just select the desired nodes and add them to the slice. A sample of this interface is illustrated in the Figure 2.3.



22 nodes currently in *dcetut_Multipath_P2P*

1 2

20 items/page

Search and

HOSTNAME	AU	ST	RES	BU	BW	CC	CF	DF	LCY	MS	OS	R	
pl1.bit.uoit.ca	PLC	boot		n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0	<input type="checkbox"/>
onelab3.info.ucl.ac.be	PLE	boot		n/a	n/a	n/a	n/a	n/a	Louvain	n/a	f12	0	<input type="checkbox"/>
planetlab1.utt.fr	PLE	boot		n/a	n/a	n/a	n/a	n/a	Troyes	n/a	f12	0	<input type="checkbox"/>
planetlab02.ethz.ch	PLE	boot...		n/a	n/a	n/a	n/a	n/a	n/a	n/a	f12	0	<input type="checkbox"/>
plab1.create-net.org	PLE	boot		n/a	n/a	n/a	n/a	n/a	Rome	n/a	f12	0	<input type="checkbox"/>
planetlab1.di.unito.it	PLE	boot...		n/a	n/a	n/a	n/a	n/a	Turin	n/a	f12	0	<input type="checkbox"/>
planetlab2.cs.vu.nl	PLE	boot		n/a	n/a	n/a	n/a	n/a	Amsterdam	n/a	f12	0	<input type="checkbox"/>
planetlab1.mta.ac.il	PLE	boot...		n/a	n/a	n/a	n/a	n/a	n/a	n/a	f12	0	<input type="checkbox"/>
planetlab-node1.it-sudparis.eu	PLE	failboot		n/a	n/a	n/a	n/a	n/a	n/a	n/a	f12	0	<input type="checkbox"/>
host4-plb.loria.fr	PLE	boot...		n/a	n/a	n/a	n/a	n/a	Nancy	n/a	f18	0	<input type="checkbox"/>
planetlab-1.research.netlab.hut.fi	PLE	boot		n/a	n/a	n/a	n/a	n/a	Helsinki	n/a	f12	0	<input type="checkbox"/>
plab2.create-net.org	PLE	boot...		n/a	n/a	n/a	n/a	n/a	Rome	n/a	f12	0	<input type="checkbox"/>
ple4.ipv6.lip6.fr	PLE	boot		n/a	n/a	n/a	n/a	n/a	n/a	n/a	f12	0	<input type="checkbox"/>
planetlab2.cs.aueb.gr	PLE	boot...		n/a	n/a	n/a	n/a	n/a	Athens	n/a	f12	0	<input type="checkbox"/>
ple1.cesnet.cz	PLE	boot		n/a	n/a	n/a	n/a	n/a	Prague	n/a	f12	0	<input type="checkbox"/>

Figure 2.3 An interface provided by PlanetLab for adding nodes.

Second method for adding nodes is to do it programmatically using the APIs that are exposed by the PlanetLab known as MyPLCAPI [12]. These APIs are based

on remote procedural call mechanism that returns an XML (Extensible Markup Language) [19] file containing the results of the API method call. Most of the current high level programming languages allow the user to use these mechanisms and the API could easily be utilized with programming languages such as Java, Python, PHP, C++ etc. This method is highly beneficial for the users when they want to dynamically control the amount of nodes or resources available to their experiments. For example an experiment based on elastic cloud computing could easily implement the property of adding resources on demand during high load periods and then scaling down as the load decreases.

After the nodes are added to the slice, the keys of the users who are assigned to the slice are propagated to the added nodes and they can start accessing the resources on each node. Depending on the number of nodes that are added to the slice on average it takes about 2-3 hours for the user's key to propagate to all the nodes according to PlanetLab but in practice this time was found to be much higher and could take up to a day for the keys to be available on all the nodes. The users can check on which nodes the keys are available through the web interface or API by querying for the nodes that have the tag `ssh_key` associated to them.

PlanetLab also provides a tool called *Sirius* that can be used for reserving the whole node for the slice for a fixed period of time. It is useful for projects that require a high amount of processing power or the measurements that might get affected by external factors such as other slivers on the node causing interference due to CPU or bandwidth sharing. This tool can give the sliver belonging to the user's slice, an increased CPU priority and allocate higher amount of bandwidth for a maximum of 30 minutes period at a time.

Once the nodes are added to the slice and the user keys are installed on the selected nodes, the users can log in to the node by using any SSH [16] client. The only difference between logging into PlanetLab nodes and other SSH servers is that the username required for the login process in SSH is actually the name of the slice and not username of the user. The login command is composed as shown below.

```
[user@host #] ssh -l slice_name -i user_private_key node_name
```


3. SYSTEM ARCHITECTURE AND DESIGN

This chapter is aimed towards describing the architecture and concept of the web-based system that was built as part of this work to provide a mechanism through which the end-users can emulate multiple connected devices on top of PlanetLab and carry out their experiments on these emulated devices.

3.1 System Concept

The developed system enabled the users to define any kind of connected device such as Smartphones, Tablets, Laptops, Sensors and Actuators etc. and model different kind of network interfaces these devices may possess. System allowed the users to model the network interfaces mainly based upon the link reliability, amount of bandwidth available and possible network latency experienced over the interface [28].

There was also the possibility for the users to define custom tags and associate these tags with the defined devices; based on these tags the users can perform lookups in the system at a later point in time.

The illustration shown in Figure 3.1 explains the underlying concept behind the system and how the system was envisioned to logically represent the emulated devices and their interconnections. In the Figure 3.1, the cloud represents the overall PlanetLab testbed and the square blocks represent the physical hosts part of the PlanetLab. On top of these hosts the system emulates the user-defined devices and communication links that the user has associated with these devices. The links maintained between these devices and how they communicate with each other are governed by the application or experiments that are run atop these emulated devices.

3.2 System Architecture

The system architecture was based on a centrally managed approach, which involves a management server as the central entity that is used to control all the aspects and

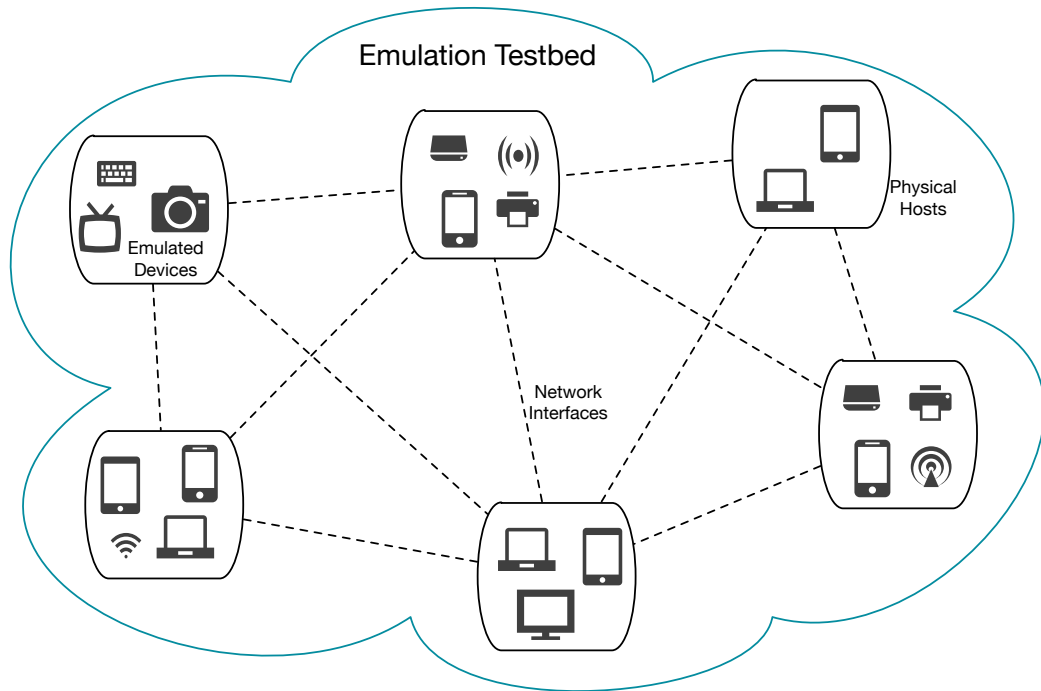


Figure 3.1 An illustration of the system concept showing the emulated devices on top of PlanetLab hosts.

functionalities provided by the system. The management server also doubled as a web-server to deliver the user interface through which the user can control and utilize the system [28].

An overview of the system architecture is presented in Figure 3.2 to illustrate the below described architecture. It shows all the major components and how these components are interconnected with each other to form the emulation platform.

The management server allowed the users to define devices and network interfaces and their characteristics through the user interface. It also generated the configuration commands as per user's directions and transmitted these configurations and details to the selected PlanetLab hosts. After the commands were executed, it reverted back to the user with the emulation devices' details such as the network addresses, ports etc. through which the user can access the emulated devices.

The central server was also in charge of maintaining the physical hosts that are part of the system. It communicated with the PlanetLab central server for obtaining a list of available physical hosts that were available for use and their details. The user can select the hosts to be included in the system based on these details presented by the server.

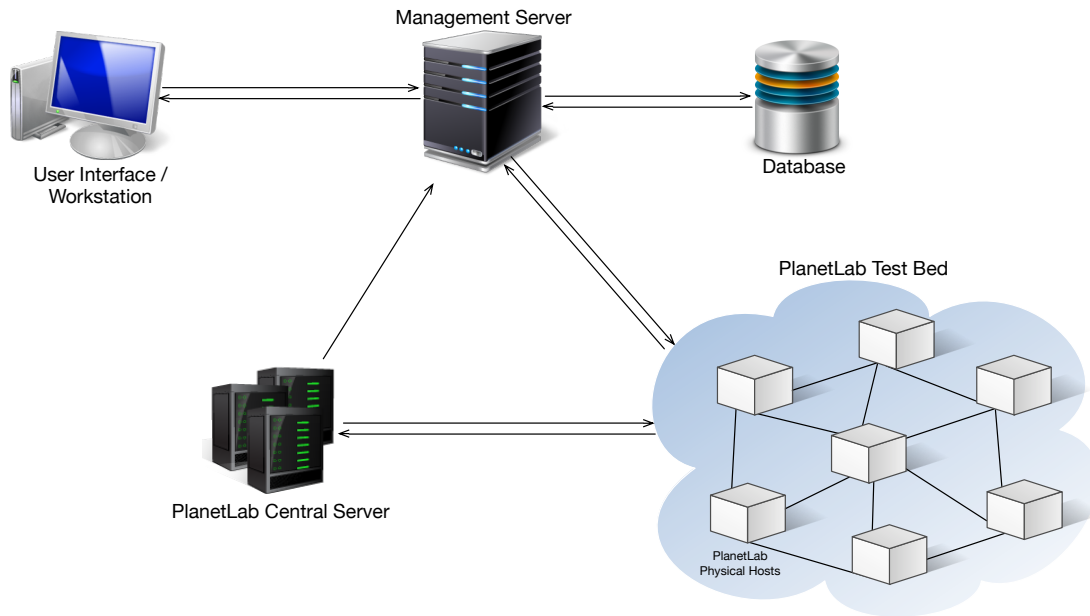


Figure 3.2 System Architecture: a design overview showing the main components and their interconnections that form the emulation platform.

There was also a database present in the system architecture that was developed as part of this work, which was maintained and utilized by the management server for maintaining a persistent state of the system.

It allowed storing and retrieving the essential data of the setup such as host information, device configurations, network links and their characteristics and the tags that were associated to these nodes and devices. This also allowed the user to re-use the already present configuration for devices and network interfaces for future experiments.

4. IMPLEMENTATION

This chapter is aimed towards describing the implementation work that was carried out for building the emulation system. It provides a detailed technical design of the system, introduces technologies, tools and programming languages used for implementation, descriptions of application flow for different system actions and finally the steps required to utilize the system along with the relevant UI screenshots and database tables.

4.1 Programming Languages and Technologies Used

The main programming language used for implementing the core processes of the system application was PHP [9] as the solution has been designed to have a web based user interface. PHP was a natural choice for programming language as it allows for easier integration of backend system functionality with the user interface that can easily be accessed using any platform with a web browser available.

This easy integration of the backend functionality with UI was made possible as PHP at runtime can generate the HTML code required for the web browser to render the UI, so the user's actions and interactions with the backend system and the result of these actions can easily be shown in real-time to the user through a web browser.

The user interface presented to the users was written with a combination of JavaScript and HTML, which allows the UI to be dynamically served to the user reflecting the state of the system without requiring the user to constantly refresh the webpage.

The database engine required for maintaining the configurations provided by the user and the state of the system was based on MySQL [8] database server. The database stored information such as the list of emulated devices and their available communication interfaces and as well as the IP Addresses and port numbers through which the user can access these devices. The language used for creating and maintaining the database was SQL [15].

The communication between the central server and the database engine was carried

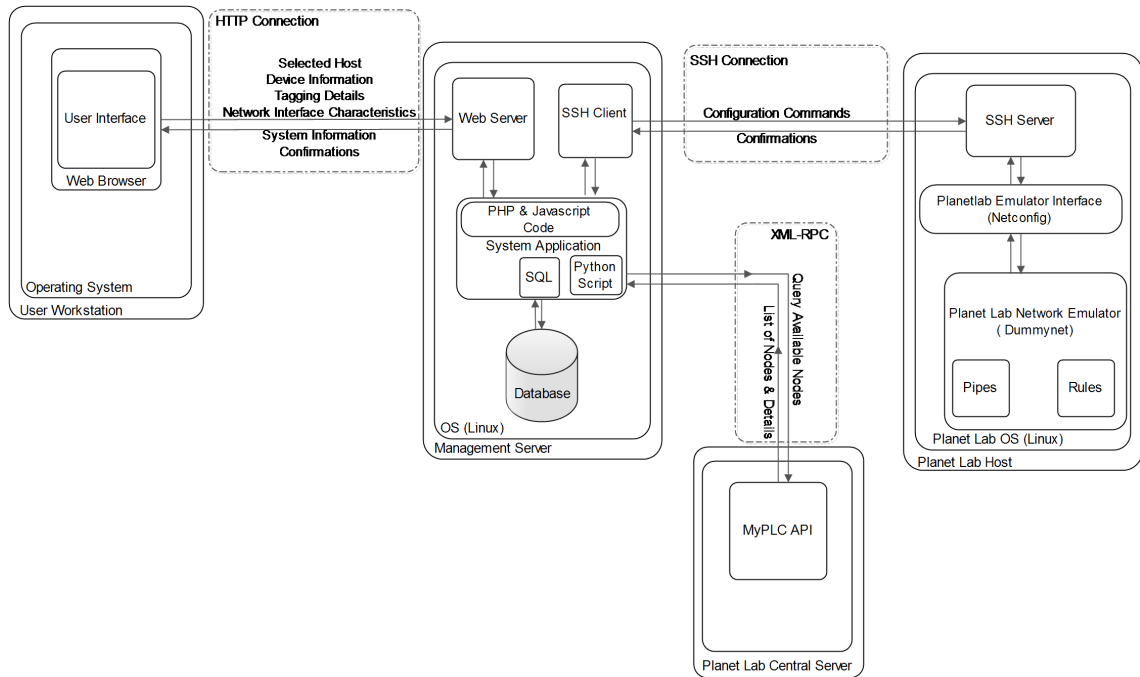


Figure 4.1 Detailed system design representing the implementation details of the system and the communication between involved components.

out using AJAX (Asynchronous JavaScript) technology, it allows for asynchronous communication to be possible between the server and the database. This enables for the dynamic update of information presented to the user through the web-based UI.

There was also a small communication module implemented in Python, which worked in conjunction with the main system application. It was in charge of communicating with the PlanetLab central servers to fetch the details of the PlanetLab nodes that were part of the logged in user's PlanetLab slice and were available to the user for running their experiments. The communication between this module and the PlanetLab central servers was based on XML-RPC (XML based remote procedure call) [20] mechanism.

4.2 Detailed System Design

A low-level detailed design to elaborate on how the solution was implemented; the main modules present in each entity involved and the type of communication between each involved component are presented in Figure 4.1.

The management server formed the core of the developed system, which was used for controlling all the other entities involved. It constituted mainly of a webserver,

SSH client, system application and a database server. Web server present in the management server allowed for the users to interact with the system application using any general web browser and the database server was used for recording and maintaining the state of the system. The database server also was utilized for storing the essential information required by the system application for emulating the configured devices and also maintaining the information for each emulated device on how to reach and communicate with the device.

The system application acted as the brain of the management server and it controlled all the components present in the system and how they interact with each other. It controlled how the UI was rendered for the user, interpreted user actions, convert the actions and configurations to commands that can be understood by the remote PlanetLab nodes, execute remotely the generated commands and manage the results of these configurations by storing them in the database server in a logical manner and present these results to the user.

The user's workstation does not require any special module other than an operating system that allows a GUI based web browser to be installed. The UI could then be accessed through the browser allowing users to manage their experiments and the emulated devices through any of the modern PCs, phones or tablets. The communication between the user's workstation and management server happened over the HTTP connection and is visually represented in Figure 4.2.

The information exchanged between the user workstation and management server was mainly about the selected host on which the device would be emulated, information regarding the emulated device and the characteristics of each network interface that would be associated with the device. Any associated tagging details to be used later for lookups and management of the emulated devices was also exchanged. The management server reverted back to the user with information representing the current state of the system and confirmation messages based on user's requests.

The communication at the back-end happened mainly between the management server and the selected PlanetLab host and as well as between the management server and the PlanetLab central server. These communications are illustrated in Figure 4.3.

The information exchanged between the management server and PlanetLab central server was related to the available physical hosts for the logged in PlanetLab user where the system application queried for the available nodes and the central server reverted with a list of available hosts and the details of each host such as the location, IP address and the architecture of the remote node. The system application used

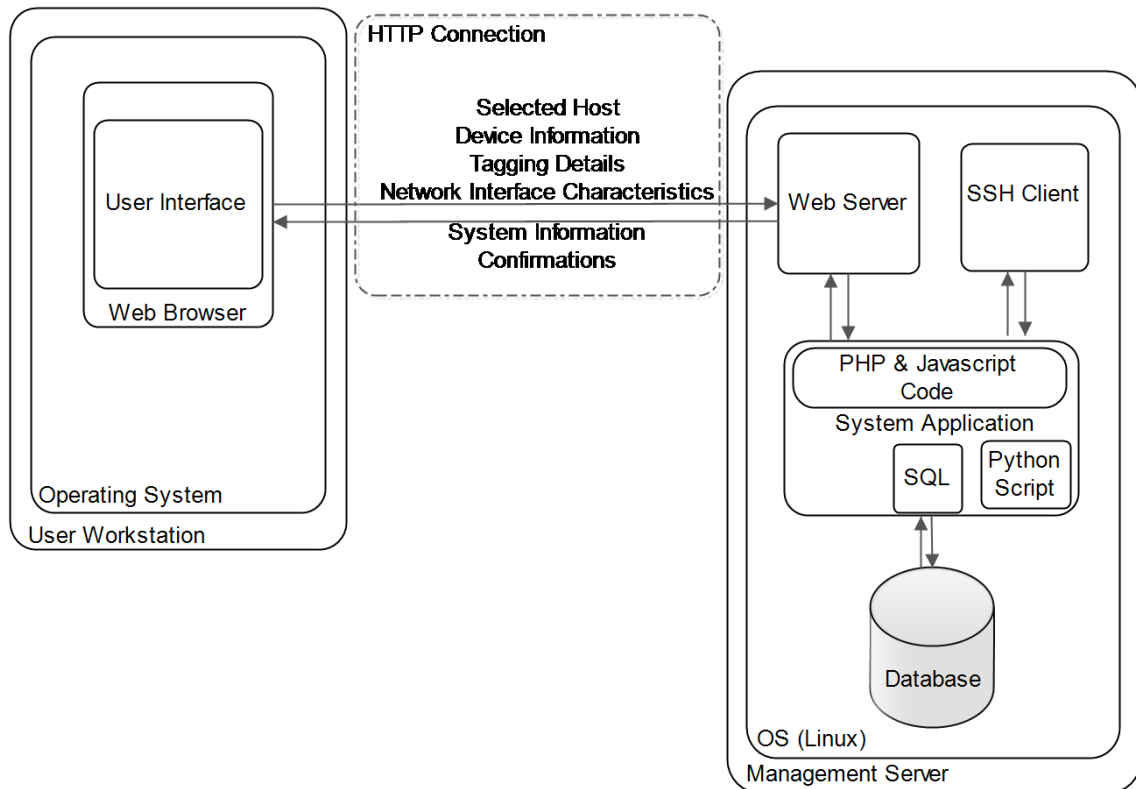


Figure 4.2 Communication between user workstation and management server.

a Python module that invoked the methods exposed by the PlanetLab API called PLC API [12]. This communication happened using XML-RPC technology and the information exchanged is encoded using XML(Extensible Markup Language) and HTTP is used as the transport protocol.

The SSH [16] client present in the management server was used for forming a secure connection to remote PlanetLab nodes i.e. the physical hosts on top which the configured device would be emulated. The information exchanged over the SSH connection was the configuration commands that were generated by the system application based on user's configurations and the confirmation messages once the requested device was successfully emulated and the details of this device through which the user could access it.

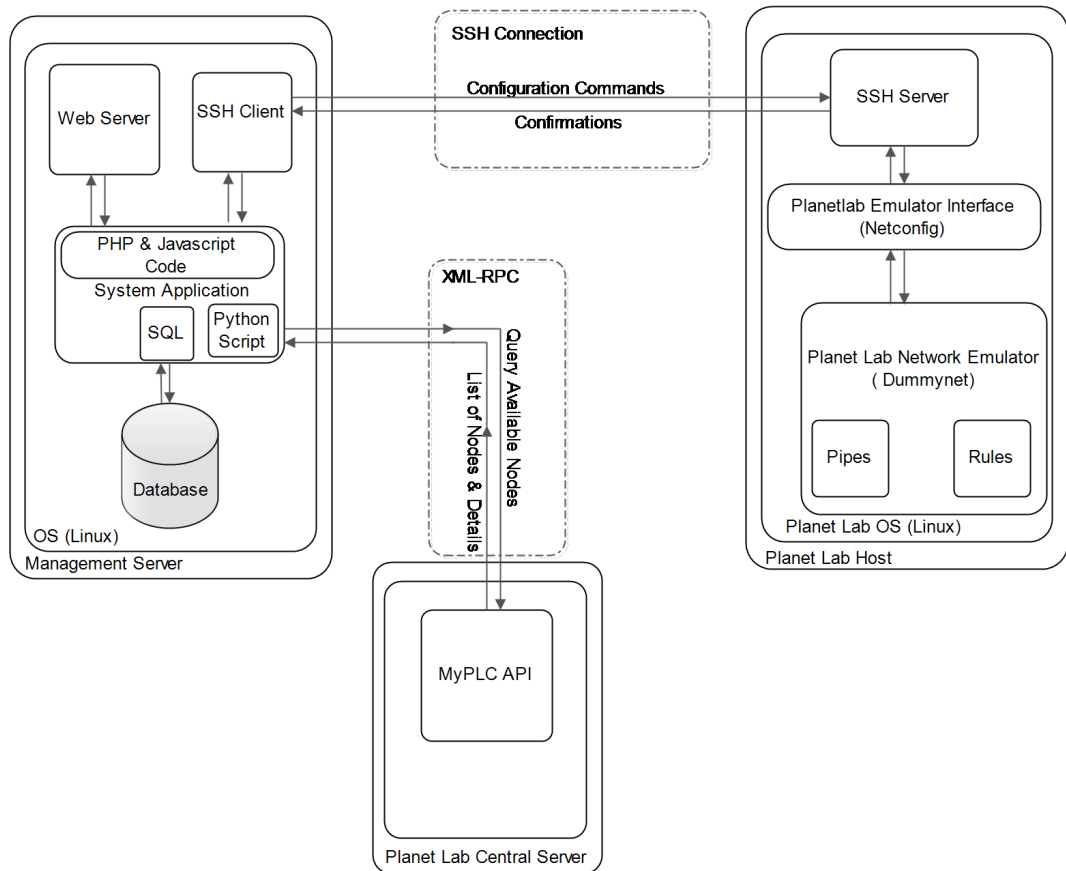


Figure 4.3 Back-end communication between the management server, PlanetLab central server and remote physical hosts

The SSH communication was authenticated by using public key cryptography. The public keys of the users were uploaded to the PlanetLab central server from where the uploaded public keys were propagated to all the remote hosts that were part of the user's PlanetLab slice. The private keys of the users were stored on the management server and were used to authenticate the server on user's behalf with the concerned remote node to which the public key of the user had already been propagated.

Emulation of the network interfaces for IoT devices was made possible on the PlanetLab physical hosts by harnessing the abilities presented by PlanetLab in the form of a command line bandwidth management tool available on the PlanetLab nodes that are added to a project's slice. The tool was based on a very simple but flexible and powerful tool for testing network protocols and topologies called Dummysnet [21]. Dummysnet was a live network emulation tool, which allowed fine grain control

over the different aspects of controlling the network traffic such as bandwidth management, latency and quality of the connection. These aspects could be modified during the run time i.e. dynamically adjusted while the communication was going on.

Dummynet was originally designed for the FreeBSD [17] operating system but since then it has been ported to work over many different operating systems. It worked by intercepting the packets at the network stack of the host operating system and utilized objects called pipes and rules for controlling the network traffic. Pipes were simple objects based on queues where the intercepted traffic was placed and a scheduler based on the configurations controlled how the packets were allowed to leave the queue. It also provided possibility of using different scheduling policy algorithms to be applied to each configured pipe object. Dummynet used ipfw (IP-Firewall) [6] for filtering the traffic on which the configured rules would be applied. Ipfw was an IP based packet filter, which was available as a kernel module that could be utilized for filtering the network traffic according to advanced rules set by the user.

When PlanetLab nodes were added to the slice being used by the project, the physical resources of the node were not directly allocated to the slice instead a portion of the resources known in PlanetLab as sliver were allocated to the slice by allocating a virtualized node to the user's project which had access only to its own share of computing resources. Due to design of the PlanetLab testbed, multiple users could have access to the same physical host and could simultaneously run their experiments over it which were totally unrelated to each other and could belong to different projects or slices in PlanetLab.

In order to cater to the simultaneous access of shared resources, security model of PlanetLab prevented the installation of network emulation tools to be installed directly on to the physical node added to the slice by the user. Allowing such network controlling tools to be freely installed would cause undesirable effects and the operation of the physical host could be affected, causing adverse effects to the other experiments that might be running on the same physical host [22].

Due to the limitations imposed by the PlanetLab security model, network emulation and bandwidth management was possible on PlanetLab nodes by utilizing a customized version of dummynet that cannot be modified or replaced by the PlanetLab users. It was exposed to individual users instead by a user-space command line interface to dummynet that let the users emulate networks only for the virtualized node assigned to their slice. This command-line tool was called netconfig [3]

and it was invoked by the system application remotely with various parameters that were auto generated based on the configurations provided by the user.

Netconfig could be configured with different roles related to the role the PlanetLab host played in the project's slice or kind of experiment it was being configured for. The three main roles available are summarized in the Table 4.1.

Table 4.1 List of roles available for network emulation using netconfig

Role	Explanation
Client	This option allowed for the node to emulate links for clients that could be used for connecting to a server on a known port or address.
Server	This option allowed the node to emulate links for servers that could be used for listening on one or more well-known ports for incoming connections.
Service	This option was a combination of the client and server mode, which allowed the emulated link to function for both clients and servers.

Along with specifying the roles for netconfig, it also allowed for the users to specify the bandwidth available on the link, time delay that should be applied to the traffic flowing over the link also the packet loss ratio i.e. the amount of packets that would be dropped over the link. The usage of netconfig is further explained by using sample configurations in different roles with different parameters.

A sample configuration command for the netconfig tool in client mode would be specified as:

```
[user@host #] netconfig config CLIENT 9434 IN bw 10Mbit/s delay
                2ms plr 0.3 OUT bw 1Mbit/s delay 5ms plr 0.1
```

The above shown command would emulate a link on port 9434 in client mode such that any incoming network packet to the node with source port set as 9434 would be throttled to 10Mbit/s bandwidth, face a delay of 2ms and as the packet loss ratio is set to 0.3, approximately 30% of the packets would be dropped before being passed on to the higher layer in the network stack. This configuration would also effect the outgoing network packets with destination port set as 9434 to a bandwidth of 1Mbit/s, a delay of 5ms and about 10% of the packets would dropped before being passed on to a lower layer in the network stack.

Configuration command example for the netconfig utility in server mode would be

given as:

```
[user@host #] netconfig config SERVER 8080 IN bw 120Mbit/s delay
                1ms plr 0.1 OUT bw 90Mbit/s delay 7ms plr 0.2
```

The command shown above would emulate a network link in server mode at port number 8080 and any incoming network packets having destination port as 8080 would be subjected to a data rate of 120Mbits/s, a delay of 1 millisecond and packet loss ratio of 10 percent. Outgoing packets having the source port as 8080 would be limited to a bandwidth of 90Mbits/s, subjected to a delay of 7ms and about 20% of such packets would be dropped.

In the emulation system created as part of this work, service mode was chosen to be the mode used for emulating the network interfaces of IoT devices. It allowed us to emulate both uplink and downlink for different network interfaces associated with different port numbers and any application data flowing over those port numbers would be subjected to the interface configurations setup by the user.

Using service mode, netconfig applies the limitations to all incoming and outgoing data packets having the destination port or the source port as the one configured for the link. A sample command for netconfig utilizing the service mode would be as shown:

```
[user@host #] netconfig config SERVICE 7183 IN bw 15Mbit/s delay
                3ms plr 0.2 OUT bw 7Mbit/s delay 1ms plr 0.1
```

The command shown above would emulate a network interface in service mode and data packets having the source port or destination port number as 7183 would be subjected to the rules specified for the emulated network interface. All incoming traffic with port number 7183 would be limited to a bandwidth of 15Mbits/s, subjected to a delay of 3 milliseconds and about 20 percent of the packets would be dropped. Outgoing traffic would be limited to a bandwidth of 7Mbits/s, a delay of 1ms and a packet loss ratio of 0.1.

4.3 System Application Flow

In this section, an overview of the system logic for emulating a device is presented. The application flow when a request comes in for emulating devices over a selected host is explained with the help of message sequence diagrams which detail the kind

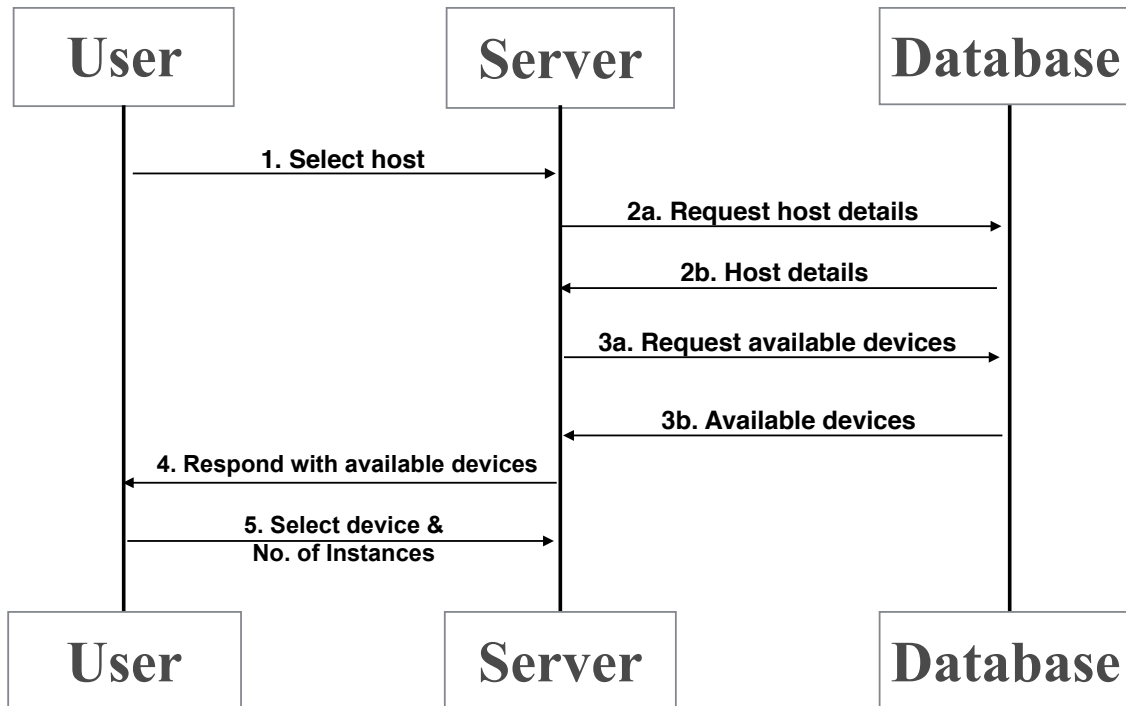


Figure 4.4 First part of the message sequence chart for system logic behind device emulation process.

of entities involved during each step and the kind of messages that are exchanged between these entities.

Before a device could be emulated on the remote host, all the required details about the device and the characteristics of its network interfaces must have been present in the system. The first step towards emulating the device was for the user to select the physical host on which the device would be emulated. Upon receiving the request from the user the system would then query the database table containing the list of available hosts and revert back to the user.

After the user had selected the host, the details for the host would be fetched from the database and also the database would be queried for a list of available devices that could be emulated on the selected host. The details of the available devices were presented to the user from which the user then selected the device and the number of instances of the device to emulate. This formed the first part of the application logic and is illustrated by the help of a message sequence chart presented in Figure 4.4.

Once the server had received the selected device and the number of instances to be emulated, it queried the database table containing the device information for the

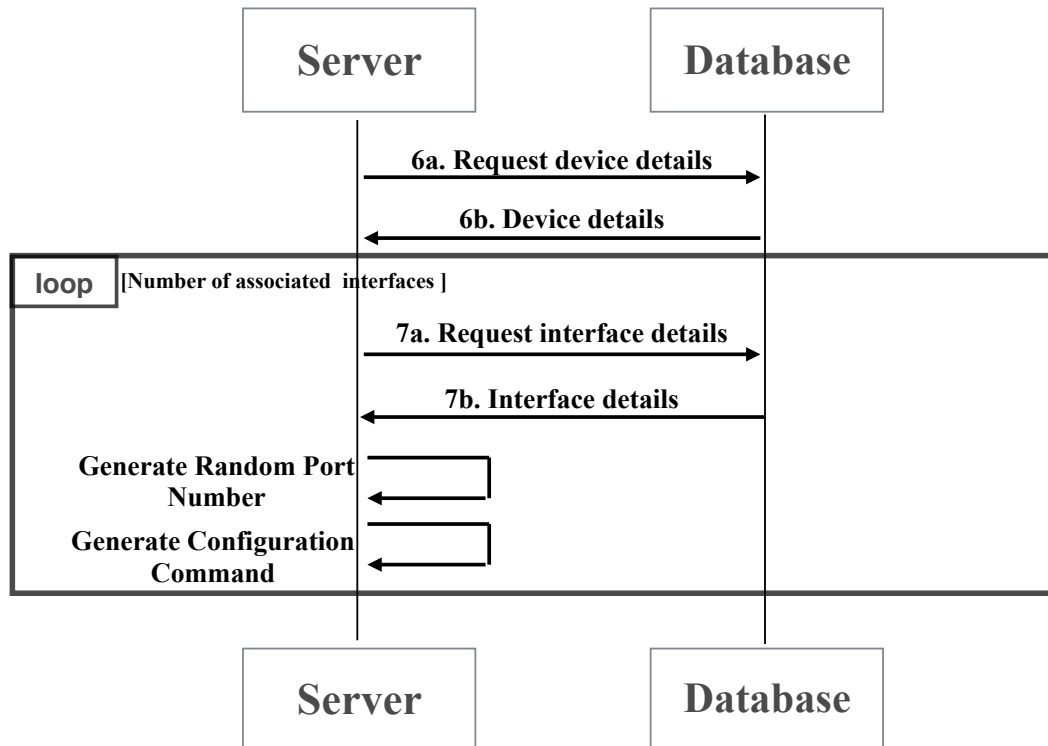


Figure 4.5 Second part of the message sequence chart for system logic behind device emulation process.

details of the device and the network interfaces that were associated with the device. Upon receiving the network interfaces, system queried the database table containing details of the network interfaces for the parameters associated with each interface that was present in the requested device.

For each interface, the system generated a random port number that was not being used on the remote host already for any other emulated link. Then based on the generated port number and the associated parameters with the interface, system generated a configuration command for the netconfig utility that would be executed on the remote host to emulate the network interface. This part of the process formed the second part of the system logic and is shown graphically in the form of a message sequence chart in Figure 4.5.

Once the configuration commands had been generated for all the network interfaces to be emulated, the system established a SSH connection to the remote host on which the device would be emulated. Upon successful establishment of the connection, system executed the generated commands for each of the network interface for the device to be emulated.

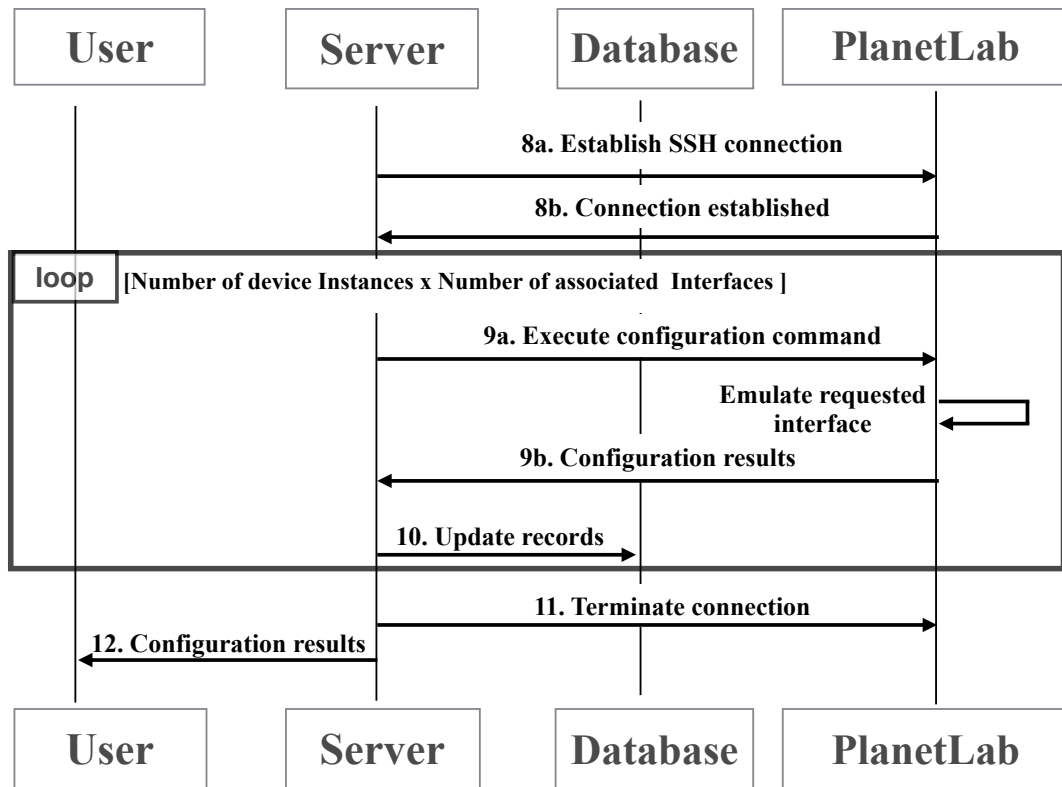


Figure 4.6 Final part of the message sequence chart for system logic behind device emulation process.

The remote host would then emulate the requested network interface and return the result as well as the details of the newly emulated interface back to the server. The system application would then update the database table containing information regarding the configured interfaces by adding the details of the new interface to the table.

This process was repeated by the system application in a loop that was executed for each of the device instance to be emulated and for all of the network interfaces that were present in the device. Once all the requested device instances had been successfully emulated the server would terminate the connection to the remote host and present the user with results of the request along with detailed information for each network interface of the emulated devices indication how they could be accessed and utilized by the user.

This process forms the final part of the system application flow behind the device emulation process and is represented by a message sequence chart between the entities involved in Figure 4.6.

The screenshot shows a web-based user interface titled "IoT Node Emulation & Management System". The main heading is "<< Add Interface >>". Below this, there are two configuration sections: "Downlink Configuration" and "Uplink Configuration".

Downlink Configuration:

- Interface Type: Bluetooth v4.0
- Bandwidth: 200 (with a spinner control) Unit: Kbit/s
- Delay: 10 (with a spinner control) Unit: ms
- Packet Loss Ratio (Plr): 0.0 (with a spinner control) 0=No Loss 1=Total Loss

Uplink Configuration:

- Bandwidth: 180 (with a spinner control) Unit: Kbit/s (selected from a dropdown menu that also shows bit/s, KByte/s, Mbit/s, and MByte/s)
- Delay: 10 (with a spinner control) Unit: ms
- Packet Loss Ratio (Plr): 0.0 (with a spinner control) 0=No Loss 1=Total Loss

At the bottom of the configuration area, there are two buttons: "Add Interface" and "Main Page".

The footer of the interface features the logo of Tampere University of Technology and the text: "TAMPERE UNIVERSITY OF TECHNOLOGY" and "Department of Pervasive Computing, Tampere University of Technology, Finland."

Figure 4.7 User interface for defining network interfaces in the system.

4.4 System Usage and User Interfaces

The user interacted with the system using a web-based user interface. Using this interface the user first needed to define the information regarding the devices and the network connections they may possess before selecting the host on which to emulate the devices. Defining tags for devices and nodes were also done through this interface and these tags could be associated to the devices and physical hosts also through this interface.

The series of steps the user needed to perform in order to utilize the system are provided in this section. The graphics representing the UI screens and the database tables used in these steps are also discussed.

4.4.1 Defining Network Interfaces

The first step the user needed to perform for utilizing the system was to define the network interfaces and their characteristics. These interfaces would then be available to the user during the next steps for associating them with the devices.

The system provided a UI for defining the interfaces shown in Figure 4.7. Using this interface, the user could define the interface type and provide configurations for

Interfaces
interface_id INT (11) - PK
interface_type VARCHAR(45)
interface_down_bw INT (11)
interface_down_bw_unit VARCHAR (45)
interface_down_delay INT (11)
interface_down_plr FLOAT
interface_up_bw INT (11)
interface_up_bw_unit VARCHAR (45)
interface_up_delay INT (11)
interface_up_plr FLOAT
configuration_data TEXT


Figure 4.8 Database table used by the system for storing information of each network interface.

both uplink and downlink characteristics of the interface. The user could provide the bandwidth in numerical digits and select the bandwidth unit from the drop down menu e.g. bit/s, Kbits/s, Mbits/s etc. It also allowed users to define the time delay experienced for uplink and downlink in milliseconds and as well as define the packet loss ratio experienced, by choosing it from the drop down menu. The value for the packet loss ratio could be defined in the range of 0.0 to 1.0 where a difference of 0.1 corresponded to a drop of approximately 10 percent packets.

Once the user added the interface and its details, the system automatically generated an identification number for the interface in order to uniquely identify the interface within the system. The system saved the information about the interface into the database relation titled *Interfaces*. This table is shown in Figure 4.8. The table contained fields for storing the individual parameters provided by the user and also a field called *configuration_data* which unified all the parameters provided into an auto generated configuration command through which the interface could be emulated when needed. The auto generated interface id acted as the primary key for this table.

The system also provided an interface containing a concise view for the user that provided a list of all the network interfaces defined in the system and their configurations. This information was presented to the user in a tabular fashion as shown in Figure 4.9. The user was also able to navigate to the editing mode for each network interface for modifying its configuration or to delete the interface from the system if required.

IoT Node Emulation & Management System							
<< Interfaces >>							
ID	Interface Name	Downlink			Uplink		
		Bandwidth	Delay	PLR	Bandwidth	Delay	PLR
18	Ethernet	100 Mbit/s	2 ms	0	80 Mbit/s	0 ms	0
19	WiFi	25 Mbit/s	4 ms	0.2	20 Mbit/s	4 ms	0.1
20	Bluetooth	700 Kbit/s	0 ms	0	700 Kbit/s	0 ms	0
21	Bluetooth v2.0	2 Mbit/s	0 ms	0	2 Mbit/s	0 ms	0
22	Bluetooth v3.0	15 Mbit/s	2 ms	0	15 Mbit/s	4 ms	0
23	Bluetooth Low Energy	200 Kbit/s	0 ms	0	200 Kbit/s	0 ms	0
24	3G	2 Mbit/s	10 ms	0.1	1 Mbit/s	5 ms	0.1
25	ZigBee	200 Kbit/s	8 ms	0	200 Kbit/s	6 ms	0



TAMPERE UNIVERSITY OF TECHNOLOGY

Department of Pervasive Computing, Tampere University of Technology, Finland.

Figure 4.9 User interface for providing a list of all defined network interfaces in the system.

4.4.2 Defining Devices & Associating Interfaces

After the user had defined the required network interfaces, the next step in the process was to define the actual device and provide the device description and as well as the information regarding which network interfaces would be available on the device.

The system provided a simple interface for defining the devices in the system as shown in Figure 4.10. The user through this interface was able to provide the device type and an appropriate description for the device detailing any specifics of the device. The user was also provided a list of all the network interfaces that have been previously defined in the system, from this list the user could simply drag and drop the interfaces that would be available on the new device. There was also a possibility for dynamically searching for the network interfaces, in case there was a large amount of network interfaces present in the system.

The information regarding all the devices defined in the system was stored in the database table called *Devices*. The table is illustrated in the Figure 4.11. This table contained fields for storing the device information provided by the user and the device identification number which was automatically generated by the system for uniquely identifying each defined device in the system. This generated identification number also served as the primary key for the *Devices* table.

IoT Node Emulation & Management System

<< Add Device >>


Device Type:

Allowed Interfaces:

3 items selected		Remove all	Add all
WiFi	-	Bluetooth	+
3G	-	Ethernet	+
Bluetooth Low Energy	-	Bluetooth v2.0	+
		Bluetooth v3.0	+
		ZigBee	+

Device Description:

Apple iPad with Bluetooth 4.0



TAMPERE UNIVERSITY OF TECHNOLOGY

Department of Pervasive Computing, Tampere University of Technology, Finland.

Figure 4.10 User interface for defining a new device in the system and associating the network interfaces to the device.

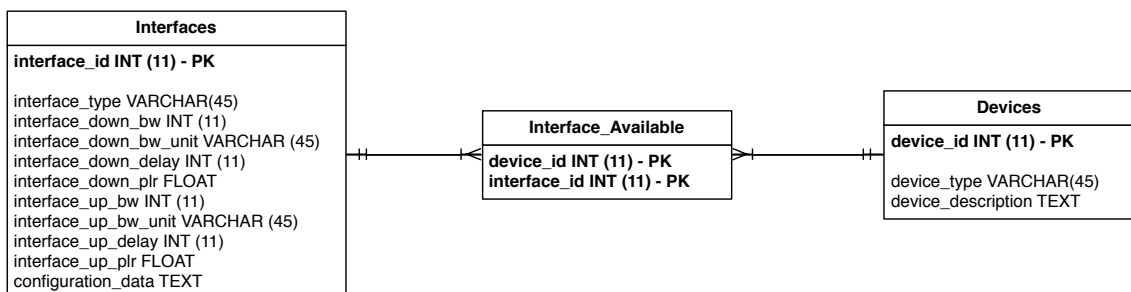
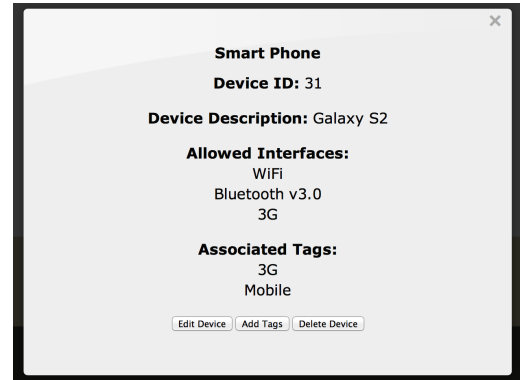


Figure 4.11 Database tables for storing information about devices and their associated network interfaces.

In Figure 4.11, the table titled *Interface_Available* used for associating interfaces with the devices is also presented. It contained only two fields, which were the device id and the interface id. For each interface present in the device the table contained a row with the id of that interface and the device id it was assigned to. The system then just had to query this table with the id of the device to find out all the network interfaces that the user had defined to be present in the device. The system also utilized this table for obtaining a list of all devices on which a particular network

Device Name	Device Description
Smart Phone	Galaxy S2
Tablet	Nexus 10
Smart Phone 2	iPhone 4S
Arduino Uno	Resource Constrained Device
Arduino Mega	Low Power RFID Reader
Remote Health Monitor	Vendor-Specific
Notebook	Generic Laptop

(a) A list of all available devices in the system.



(b) Details of a device and the available operations.

Figure 4.12 User Interface for listing all the defined devices and their details.

interface is present by just querying the table with id of the network interface.

The system application also provided an interface where the user could see a summary of all the available devices that have been previously defined in the system. An example of this user interface is represented in the Figure 4.12(a). Using this interface the user could query for device details which provided a detailed summary of the device indicating the available interfaces, device description and also any tags that were associated with the device. This interface is presented in Figure 4.12(b), using this interface the user was also able to edit or delete the device when not needed anymore.

4.4.3 Adding and Managing the Physical Nodes

The next step after defining the devices and their network capabilities was to add the physical nodes on which the devices would be emulated to the system. The user needed to setup the PlanetLab project slice and his credentials for the PlanetLab into the system application before adding nodes to the system.

The prerequisite for the users before this step was to create and setup the project slice in PlanetLab before they could use the emulation system for their experiments. The creation and setup of PlanetLab slices was performed using the interface exposed by the PlanetLab through its web-interface.

Once the project slice had been setup, the system was able to fetch the list of available hosts for the user's slice that could be used for emulating the IoT devices. These nodes were presented to the user through the user interface for adding nodes

as a drop down list. The user was able to select the nodes from this list and as well as provide custom names for these nodes through the interface. The provided name by the user was then used to refer to the node throughout the system. This interface is represented in the Figure 4.13.

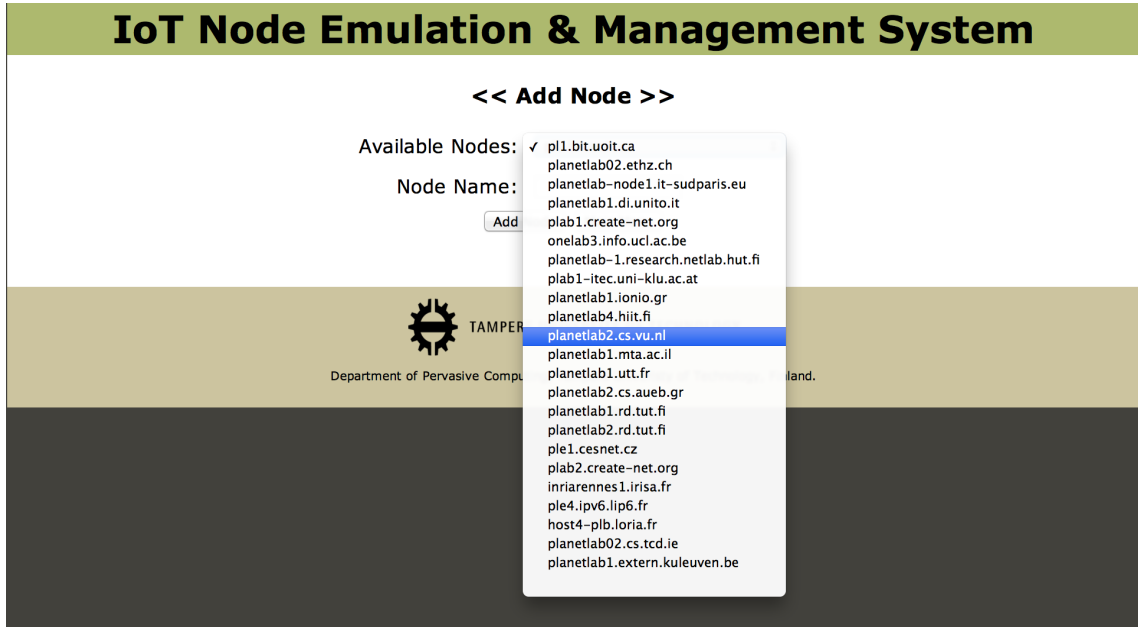


Figure 4.13 User interface for adding physical nodes to the system.

The added nodes and their details were stored by the system in the database table called *Nodes*. The system application automatically generated an identification number for each added node, which was then used internally for uniquely identifying the particular node. This identification number also served as the primary key for the database table *Nodes*.

The table contained fields for storing the node identification number, the name provided by the user and as well as the IP address of the node through which the system could contact the node. The IP address was automatically fetched by the system using its hostname from the PlanetLab central servers. The table is graphically illustrated in the Figure 4.14.

The system was also able to keep track of all the network interfaces and devices that had been emulated over the host and if these configurations were currently active. The system also stored information regarding the date and time at which the device had been configured over the node. This information was stored in the database table titled *Configurations* shown in Figure 4.14.

The system was automatically able to query the remote node about the active con-

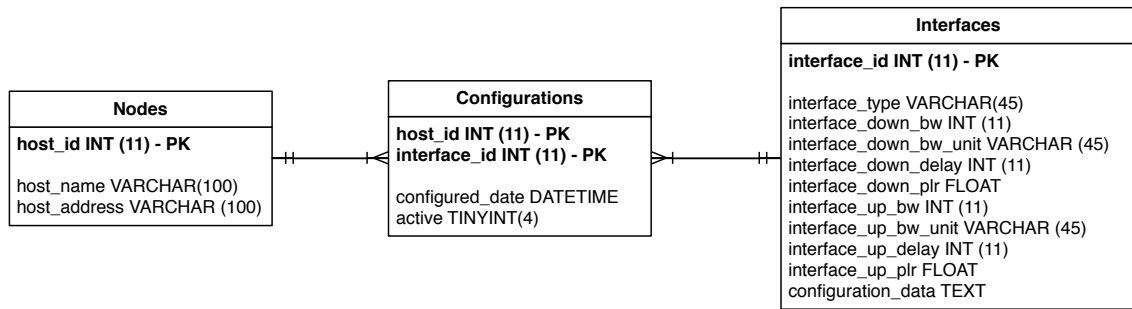


Figure 4.14 Database tables for storing information about nodes and their associated device configurations.

figurations and update the database accordingly. This kind of active tracking of configurations was necessary as the PlanetLab host according to their security policy cleared all the configurations related to network emulation automatically every 10 to 12 hours. In order for the user's emulations to be persistent and to avoid any adverse affects on the user's experiments, the system application automatically renewed the configurations before the PlanetLab nodes cleared them. This functionality was enabled by default unless the user specifically disabled it.

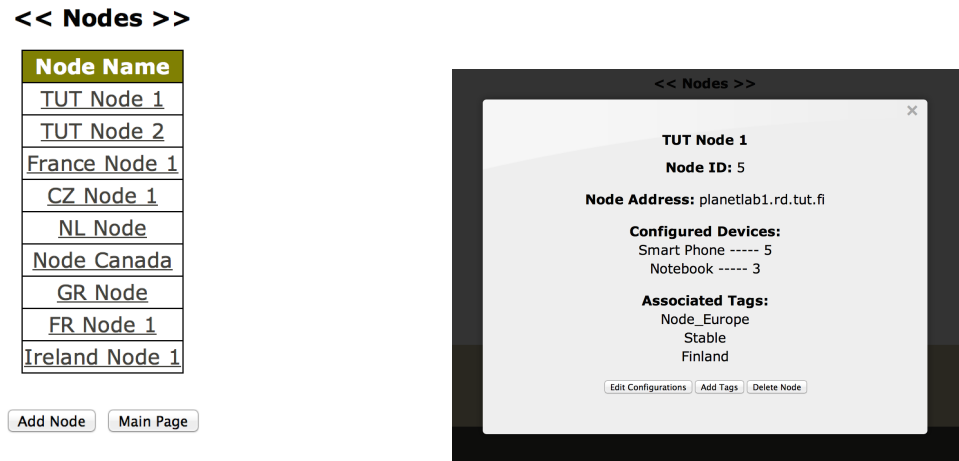
The system application also provided an user interface for listing all the available nodes that have been previously added as shown in Figure 4.15(a). The user was also able to query for a detailed summary of each of the nodes. This summary provided details of the node's address, the given name, identification key and any associated tags. The user was also able to see a detailed view of all the devices and the number of instances of each device that had been emulated on that particular node. This summary interface is represented in the Figure 4.15(b).

The user was also able to modify the details of the node or edit any associated configurations through this interface. It provided support for removing or adding tags to the node and the node could also be deleted from the system once it was no longer required from this interface.

4.4.4 Emulating and Managing the Devices

Once the devices had been defined and the PlanetLab nodes added to the system, the next step in the process was to select the node and the device to emulate for the emulation process to begin. The users had to navigate to the node on which they wanted to emulate and select the *Configure Devices* option.

The *Configure Devices* interface provided the users with a drop down list of all the



(a) A list of all available nodes in the system.

(b) Detailed summary of a node and its available operations.

Figure 4.15 User Interface for listing all the added nodes and their details.

available devices from which the user could select the device to emulate. The next step was to enter the number of instances of the device the user wanted to emulate and click on the *configure* button for the emulation process to begin. This interface is represented in Figure 4.16.

IoT Node Emulation & Management System

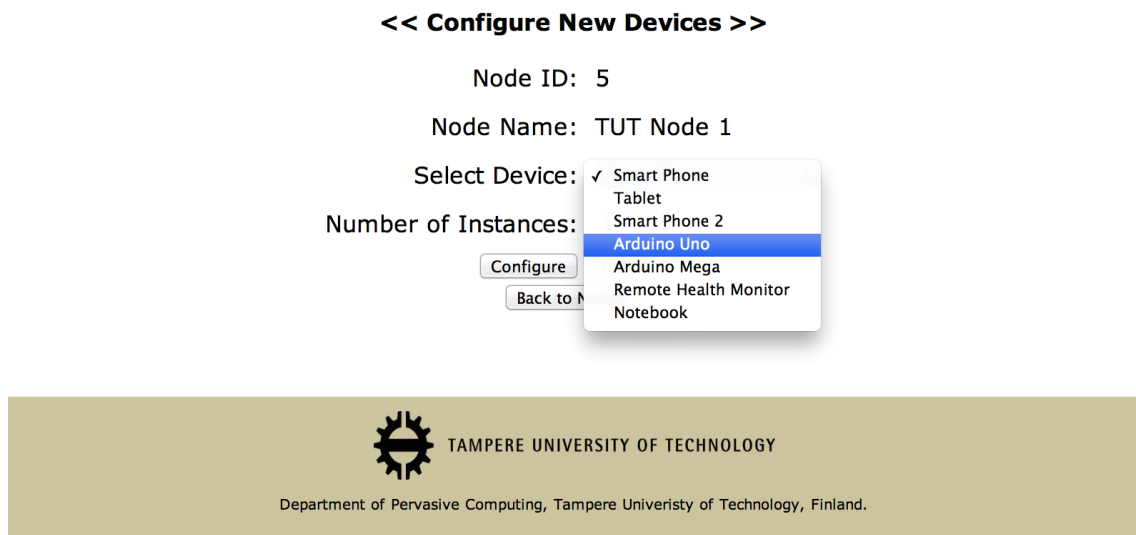


Figure 4.16 User interface for emulating the defined devices over PlanetLab nodes.

During the device emulation process, the system would automatically find the unused port numbers that could be used on the node for emulation and augment them to the configuration commands fetched from different database tables. After the

commands were ready, the system would execute these remotely on the selected PlanetLab node and add this information to the database upon confirmation from the remote nodes.

The database tables used for storing the information about emulated devices are illustrated in the Figure 4.17. Each instance of the emulated device was stored as an entry into the database table entitled *Device_Instances* which mapped the device instance to the node on which it was emulated and the type of device the instance referred to by storing the id of the node and the id of the device. Each device instance was also assigned an automatically generated identification key for identifying it within the system logic. This identification key was also stored in the same table and acted as the primary key for the table *Device_Instances*.

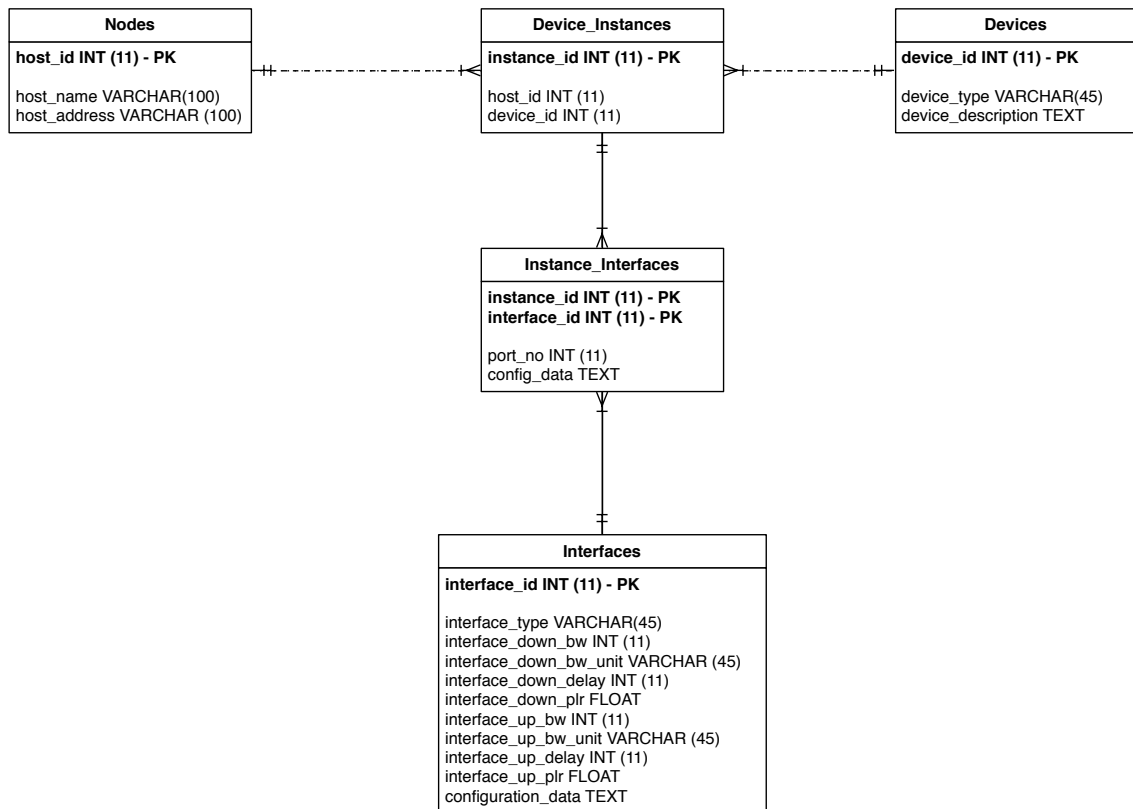


Figure 4.17 Database tables for storing information about emulated device instances and their associated configurations.

The system also maintained a table entitled *Instance_Interfaces* represented in the Figure 4.17. This table was used for storing information about each instance of the interfaces that were associated with the device instance being emulated. This table maintained the association between the interface instance and the device instance by storing the identification key of the network interface and the identification key of the device instance.

These keys were combined together to form the primary key of *Instance_Interfaces* table in which each entry corresponded to an emulated interface in the system and the combined keys could be used for uniquely identifying each interface. The table also contained a field for storing the port number of each interface i.e. the port number at which the emulated interface was available for communicating.

The field called *config_data* was used for storing the complete command that could be used for emulating the interface. This command was used in the scenario where the system application needed to refresh the configurations of the interface, if it was removed by the PlanetLab host before the user had explicitly asked to delete the device instance.

A summary of all the devices emulated in the system was made available to the users through an interface shown in Figure 4.18. The interface provided the users with a list of configured devices on each node. It also showed the amount of emulated instances of the device and the real-time status of the device, which indicated if the device was ready for communication, or not.

IoT Node Emulation & Management System

System Configurations

Node Name	Configured Device	No.of Instancs	Status	Details
TUT Node 1	Smart Phone	5	OK	Detail
TUT Node 1	Notebook	3	OK	Detail
TUT Node 2	Tablet	3	OK	Detail
TUT Node 2	Smart Phone 2	10	OK	Detail
CZ Node 1	Arduino Uno	2	OK	Detail
CZ Node 1	Arduino Mega	25	OK	Detail
FR Node 1	Remote Health Monitor	15	OK	Detail

[Nodes](#) [Devices](#) [Interfaces](#) [Device Tags](#) [Node Tags](#)



TAMPERE UNIVERSITY OF TECHNOLOGY

Department of Pervasive Computing, Tampere Univeristy of Technology, Finland.

Figure 4.18 User interface for showing the list of all emulated devices in the system.

The users were able to query the details of each device by clicking the *Detail* button. It provided users a view with the details of the emulated device and the available interfaces on the emulated device. Each emulated interface was presented in a tabular manner indicating the details of the interface. This user interface is represented

in the Figure 4.19.

IoT Node Emulation & Management System

<< Device Emulation Details >>

Node ID: 5

Node Name: TUT Node 1

Node Address: planetlab1.rd.tut.fi

Configured Device: Smart Phone

Number of Instances: 5

Available Interfaces

WiFi

Bluetooth v3.0

3G

Interface Details

Device Instance ID	Interface ID	Interface Name	Port #	Interface Address	Status
162	19	WiFi	62729	193.166.167.4:62729	Online
162	22	Bluetooth v3.0	20854	193.166.167.4:20854	Online
162	24	3G	8835	193.166.167.4:8835	Online
163	19	WiFi	8048	193.166.167.4:8048	Online
163	22	Bluetooth v3.0	44327	193.166.167.4:44327	Online
163	24	3G	6006	193.166.167.4:6006	Online
164	19	WiFi	23186	193.166.167.4:23186	Online

Figure 4.19 User interface for showing a detailed summary of an emulated device in the system.

The user was provided with the port numbers and the IP address of each interface and the status of the network interface indicating if the interface was online and ready for use. Using this information the user could utilize the interfaces of the device by configuring the application or experiment to use the address and port number corresponding to the interface for network communication. Any traffic then sent to or from this address was subjected to the configured network parameters of the interface.

4.4.5 Adding Tagging Information to Devices and Hosts

The system allowed the users to add tagging details to the defined devices and the added PlanetLab hosts. Using these tags the devices and the nodes could be grouped together according to their related properties and characteristics. These groupings


IoT Node Emulation & Management System

<< Add Device Tag >>

Tag Name:

Tag Description:

IoT devices with capabilities of receiving media streams from other connected devices



TAMPERE UNIVERSITY OF TECHNOLOGY
 Department of Pervasive Computing, Tampere University of Technology, Finland.

(a) User interface for defining new tags in the system.

IoT Node Emulation & Management System

<< Device Tags >>


Tag Name	Description
Low-Power_Sensor	Devices with low power consumption
Health	Healthcare Devices
Monitor	Monitoring Nodes
3G	Devices with 3g connectivity
Mobile	Mobile Nodes
Fixed	Nodes with fixed positions


TAMPERE UNIVERSITY OF TECHNOLOGY
Department of Pervasive Computing, Tampere University of Technology, Finland.

IoT Node Emulation & Management System

<< Node Tags >>

Tag Name	Description
Node_Europe	PlanetLab node in EU
Stable	Nodes with relatively high uptime
Unstable	Nodes with relatively high downtime
Finland	Nodes in Finland
Node_France	Planet Lab Nodes in France
Node_Germany	PlanetLab nodes in Germany
Non_EU	PlanetLab nodes outside EU


TAMPERE UNIVERSITY OF TECHNOLOGY
Department of Pervasive Computing, Tampere University of Technology, Finland.

(b) A summary of all the defined device tags.

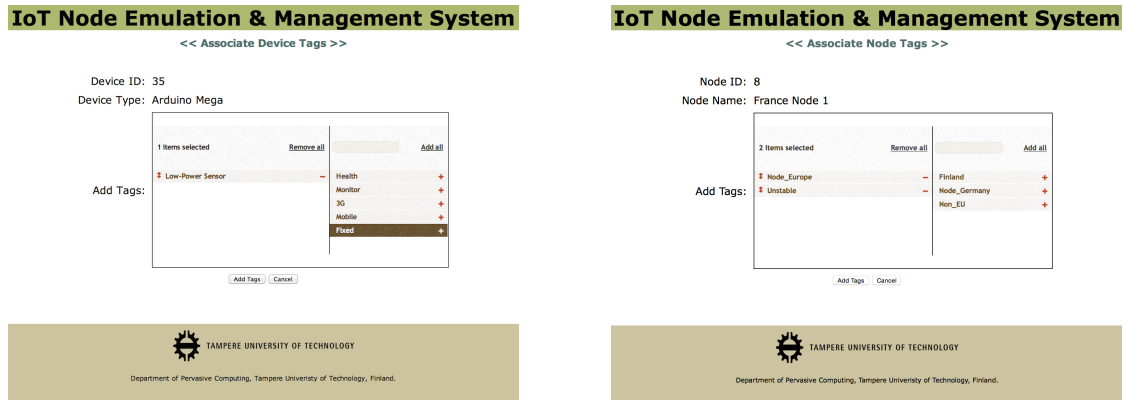
(c) A summary of all the defined node tags.

Figure 4.20 User Interfaces for adding & listing all the defined tags and their descriptions.

were useful for the users when performing lookups especially in the case when there was a large amount of devices and hosts defined through the system.

The system application provided users with the option for utilizing two types of tags. *Node Tags* were defined for the PlanetLab nodes and could be associated to the nodes that had been added. Similarly the *Device Tags* were defined for the devices and they could be associated to the devices that had been already created.

Users were first required to add the tags and their associated descriptions to the system through the provided interfaces. The user interface for adding tags to the system is shown in Figure 4.20(a). Users were also provided with interfaces that listed and summarized all the available tags in the system. Example user interface showing a list of all defined devices tags in the system is presented in Figure 4.20(b) and the interface listing all the available node tags is shown in Figure 4.20(c). From these summary views, the users were able to view the details of each tag in the system and access the edit mode for these tags. In the edit mode the users were able to modify the details of the defined tag or delete the tag from the database when it was no longer required.



(a) User interface for associating tags to the devices.

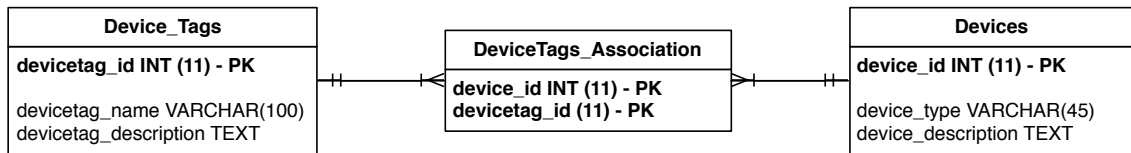
(b) User interface for associating tags to the nodes.

Figure 4.21 User Interfaces for associating device tags and node tags.

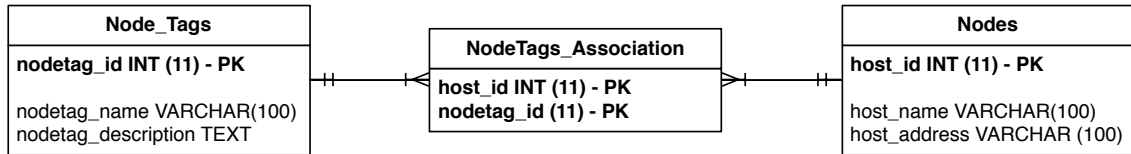
Once the tags were defined, the users could then associate these tags to the devices and hosts. Tags could be associated to the devices and hosts using the interface that listed the details of the devices and nodes as shown in Figure 4.12(b) and Figure 4.15(b). Users were presented with an intuitive interface for adding the tags by providing all the available tags in the system as a list from which the users could just drag and drop the required tags to assign them to a particular device or node. An example of the user interface for adding the tags to a device is shown in Figure 4.21(a) and an example of the user interface for adding node tags is represented in Figure 4.21(b). Search functionality was also provided to the user through this interface, which allowed the users to dynamically search for the required tags in case of a large amount of tags defined in the system.

Tags added to the system were stored in the database for persistent storage. When the users added tags, they were automatically assigned an identification key for unique identification of these tags in the system application logic. The device tags were stored in the database relation entitled *Device_Tags*. This table contained information about the user space naming of the tag and as well as the system generated id which also served as the primary key for the table. Description of each device tag was also stored in the same table. A graphical representation of the table is presented in the Figure 4.22(a).

The information about the node tags was stored in a separate database table called *Node_Tags* which similar to the table for device tags stored information regarding the given title of the tag and as well as its description. The system generated *devicetag_id* served as the primary key for the table and an illustration of the table is presented in the Figure 4.22(b).



(a) Database tables for storing information about device tags and their associations.



(b) Database tables for storing information about node tags and their associations.

Figure 4.22 Database tables for storing information about tags and their associations in the system.

In the system logic, the associations of tags to the nodes and devices were handled by the database tables called *NodeTags_Association* and *DeviceTags_Association* respectively. These tables maintained the associations by storing the node or device id and the tag id in the same row as shown in Figures 4.22(a) & 4.22(b).

These tables were also utilized by the system to resolve user's search queries based on tags. If the user queried for example a list of all devices that had a particular tag associated to it, the system would first look up the id for the tag in question and then query the association table for all the rows that contained that tag id which would allow the system to easily find the identification keys of all the devices associated with the tag. Using these keys the system was then able to return the details of each associated device back to the user.

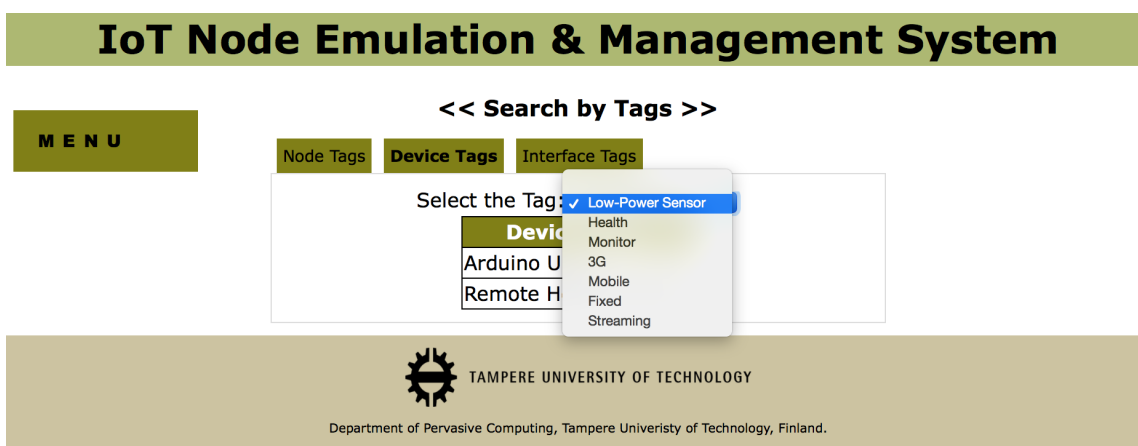


Figure 4.23 User interface for providing the search functionality based on associated tags.

A search interface was provided to the users for performing lookups in the system based on tags. System provided searching functionality based on device tags, node tags and search based on available network interfaces in the device. The interface allowed users to select the type of tag they want to search with and based on users selection a list of all tags of that type was provided to the user in the form of a drop down menu. The user then could just select the tag of interest from this list and the system would dynamically return the results of the search back to the user in the same interface. An example of the search interface is presented in the Figure 4.23.

5. RESULTS AND FUTURE WORK

This chapter sheds light on the results that were achieved as part of this work. It provides a set of measurements that were taken using the emulated nodes to verify the emulated devices were conforming to the configured parameters. A brief discussion regarding the feasibility of the approach and future work to enhance the system is also provided. Finally a concise summary of the work completed over the course of this thesis is presented in the conclusion section.

5.1 Verification and Results

This section provides a summary of the verification process and the measurements taken while utilizing the emulated devices with various configuration parameters. The first thing to be verified was that the system was generating correct configuration commands based on user's input and these commands were in accordance to the guidelines provided by the PlanetLab's netconfig utility. Secondly, it needed to be verified if the management system was establishing the connection to the chosen PlanetLab host and executing the configuration commands successfully. Final verification step was to observe the traffic flowing through the network interfaces of the emulated devices, and verify that it was conforming to the parameters provided for the interface it was utilizing.

In order to verify that the generated commands by the system application were conforming to valid netconfig utility commands, a regular expression matching tool provided as part of the PHP language called *preg_match* [10] was utilized. The system was provided with a regular expression that was designed for checking the validity of the generated string. After generating every new command the system would use the *preg_match* method and test the generated string against the regular expression and only in the case it passed the test, it would store the command in database and move forward with the next steps.

Second step in the verification process for checking that correct commands were being successfully executed over the selected host was carried out manually. After the system responded with confirmation regarding the status of emulated device, a

remote SSH connection was manually established and the command for showing the set rules and pipes provided by netconfig was used to compare the configured rules to the parameters provided by the user. This step was repeated for several kinds of devices over various different nodes to make sure the system application always configures the correct parameters and only then shows a confirmation message to the user. The commands used for listing all the pipes and configured rules in netconfig are shown below.

```
[user@host:] netconfig show -all rules
[user@host:] netconfig show -all pipes
```

After the previous verifications were completed, the final part to verify that all the traffic flowing through these emulated interfaces conformed to the parameters set forth by the user, a network measurement tool was used. This network measurement tool was called *Iperf* [5] and it was capable of observing traffic in both client and as well as the server mode. It allowed to create TCP [27] and UDP [33] stream between two specified hosts and provided with throughput measurements for the underlying connection.

```
[dcetut_Multipath_P2P@planetlab2 ~]$ iperf -c planetlab1.rd.tut.fi -p 5345
-----
Client connecting to planetlab1.rd.tut.fi, TCP port 5345
TCP window size: 16.0 KByte (default)
-----
[ 3] local 193.166.167.5 port 53143 connected with 193.166.167.4 port 5345
[ ID] Interval      Transfer    Bandwidth
[ 3]  0.0-10.3 sec  6.12 MBytes  5.00 Mbits/sec
```

Figure 5.1 Measurements showing the observed bitrate for outgoing traffic over the emulated interface.

For testing purposes a test device was created in the system, which had a single network interface associated. This interface was configured to have an uplink bandwidth of 5 Mbit/s and a downlink bandwidth of 5 Mbit/s as well. This device was deployed over a PlanetLab node and Iperf was installed on this node. The bandwidth was measured for the traffic flowing through the port number which was associated with emulated interface to verify if it was conforming to the configured parameters.

Figure 5.1 shows the measurements observed for the outgoing traffic and Figure 5.2 shows the measurements observed for the incoming traffic. It can be seen from these figures that the bandwidth observed on the interface was the same as the one that it was configured for by the system application.

The next step in the verification process was to verify if the time delay setup on the

```
[dacetut_Multipath_P2P@planetlab2 ~]$ iperf -s -p 5345
-----
Server listening on TCP port 5345
TCP window size: 85.3 KByte (default)
-----
[ 4] local 193.166.167.5 port 5345 connected with 193.166.167.4 port 40390
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.0-10.6 sec  6.12 MBytes  4.83 Mbits/sec
```

Figure 5.2 Measurements showing the observed bitrate for incoming traffic over the emulated interface.

network interface was correctly being applied. In order to verify it, a time delay of 50ms was added to the configuration parameters of the interface. Once the system application had successfully shown the confirmation, the network measurements were taken again. The results of these measurements are presented in Figure 5.3. It can be seen from the observed results that the time delay on the traffic flowing over the network interface was indeed being applied and this resulted in lower bandwidth being observed.

```
[dacetut_Multipath_P2P@planetlab2 ~]$ iperf -s -p 5345
-----
Server listening on TCP port 5345
TCP window size: 85.3 KByte (default)
-----
[ 4] local 193.166.167.5 port 5345 connected with 193.166.167.4 port 43349
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.0-10.6 sec  5.62 MBytes  4.45 Mbits/sec
```

Figure 5.3 Measurements showing the observed bitrate for traffic over the emulated interface with a forced time delay.

The final step in the process was to verify that the packet loss ratio configured on the network interface was being followed. In order to verify it, a packet loss ratio of 0.1 was setup on the network interface i.e. approximately 10% of the packets would be randomly dropped on the interface. After configuring the interface and upon receiving the confirmation from the system application, a new set of measurements were taken to observe the effects of the packet loss ratio. The results obtained from the measurement are provided in the Figure 5.4. It can be seen from the results that the packet loss ratio indeed resulted in the packets being dropped on the interface, which caused a drop in the observed bandwidth.

As observed from the set of measurements that were taken, the emulated devices and their interfaces were being correctly setup on the target nodes and the traffic flowing on the emulated interfaces was conforming to the settings provided by the user.


```
[dcetut_Multipath_P2P@planetlab2 ~]$ iperf -s -p 5345
-----
Server listening on TCP port 5345
TCP window size: 85.3 KByte (default)
-----
[ 4] local 193.166.167.5 port 5345 connected with 193.166.167.4 port 58947
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.0-10.9 sec  1.62 MBytes  1.25 Mbits/sec
```

Figure 5.4 Measurements showing the observed bitrate for traffic over the emulated interface with a packet loss ratio of 0.1.

5.2 Future Work

The solution developed as part of this work enabled the users to successfully issue commands to the PlanetLab nodes and model the emulated devices along with their interface and link characteristics. This kind of emulation was especially useful in the scenarios involving pairwise interactions but the solution would prove much more useful in the case where other aspects of the IoT devices could be emulated as well.

Modeling characteristics such as different execution or processor speeds would be highly beneficial. It would allow the users to conduct experiments and study the IoT related problems from the point of view of having constrained processing resources as well. Similarly different processor architectures could be emulated as well which would allow the emulated devices to exhibit a behavior very close to the physical devices. Finally storage requirement emulation along with the processor architecture and processing speeds would allow the emulated devices to depict very closely the nature of physical connected devices from many different aspects.

This kind of emulation still remains a big challenge to achieve with the current resources and tools available through PlanetLab. The physical machines available through PlanetLab are highly homogeneous both in terms of the hardware available and the operating system running atop these nodes. All of the nodes typically run on x86-based architecture. However it is highly possible that in the future, other projects or even PlanetLab might take on this channel and provide mean for emulating other hardware characteristics as well.

Emulating these characteristics would have an adverse effect on startup and configuration times, as PlanetLab hosts would not know in advance the kind of devices they are supposed to instantiate until the actual configuration command would be executed by the management server. In such case, it can be envisioned that an additional component might become a necessity for transferring binary images of devices to the physical hosts for emulating these devices.

One other area identified that can be improved in the future is related to architecture of the developed solution. The current architecture was aimed towards providing a single realm of control. It provided means of managing the emulated nodes through a centralized management server and the information about emulated devices and their interfaces was stored in a single database. This class of architecture is well suited for scenarios where the management is controlled by a single organization. Some example scenarios where this approach can be beneficial are small grid operator, nation-wide traffic management system and as well as smart city based management systems.

Another way to approach the architecture would be to decentralize the management and enable distributed management and node configurations by using a protocol driven approach. Currently the configuration commands were being issued using SSH protocol as blocking operations. This could be changed towards having well defined request and response messages for the instantiation and management of the emulated devices. It would imply having a management interface at each node capable of receiving the incoming messages and able to reply with messages containing various types of information about the emulated devices. While this strategy would help in distributing the management and avoiding the usage of blocking calls, it would give rise to finding solutions for managing the access control and as well as the transport layer security.

6. CONCLUSION

The main outcome of this work has been the development of a distributed platform that allowed the users to emulate network-enabled devices along with their interfaces atop the PlanetLab hosts. The emulated devices could possess multiple links of varying characteristics to simulate fixed, wireless and virtual interfaces found commonly in mobile devices such as smartphones, laptops and tablets. These actively interconnected nodes are expected to be utilized for modeling future scenarios and behavior of devices and sensors in a city or even countrywide configurations.

The developed system also provided means for remote configuration of these devices in a scalable manner and also provided tools for remotely managing the deployed devices. It also had the option for associating tags with the emulated devices and physical hosts of the PlanetLab for categorizing them. These tags could then be used later for performing lookups and retrieving the information for a particular set of nodes and devices forming organized viewpoints of the system.

The study undertaken to accomplish the development of this solution also provided a good overview of how different testbeds are created and how they can be utilized in novel ways to experiment and study various problems related to the field of computer networking and Internet of Things. The study also enabled to gain a deep understanding of the PlanetLab testbed and the various methods on how it can be utilized.

The results achieved through utilizing the developed system suggests that the approach undertaken for designing the system is highly feasible for modeling device heterogeneity ranging from very simple sensors and actuators to much more powerful devices. It proves to be also very suitable for modeling the interface heterogeneity commonly found in connected devices nowadays. It allows us to model the wireless network characteristics to customize link reliability, channel throughput and as well as the bandwidth availability.

REFERENCES

- [1] Android Mobile Operating System . Available at: <https://www.android.com>.
- [2] Common Internet File System. Available at: <https://www.samba.org/cifs/>.
- [3] Emulation - PlanetLab Europe. Available at: <https://planet-lab.eu/doc/guides/user/practices/emulation>.
- [4] Fedora - A Red Hat sponsored community project. Available at: <http://www.fedoraproject.org>.
- [5] Iperf, The TCP/UDP bandwidth measurement tool. Available at: <https://www.iperf.fr>.
- [6] IPFW, IP firewall written for FreeBSD. Available at: <https://www.freebsd.org/doc/handbook/firewalls-ipfw.html>.
- [7] MyPLC - A PlanetLab central portable installation. Available at: <https://svn.planet-lab.org/wiki/MyPLCUserGuide>.
- [8] MySQL - The world's most popular open source database. Available at: <http://www.mysql.com>.
- [9] PHP, A general purpose scripting language for web. Available at: <http://www.php.net>.
- [10] PHP Manual: Text Processing. Available at: <http://www.php.net/manual/en/function.preg-match.php>.
- [11] PlanetLab - An open platform for developing, deploying and accessing planetary-scale services. Available at: <https://www.planet-lab.org>.
- [12] PlanetLab Central API Documentation. Available at: https://www.planet-lab.org/doc/plc_api.
- [13] PlanetLab Europe - A global facility for the deployment of new network services. Available at: <http://www.planet-lab.eu>.
- [14] Putty, A SSH and Telnet tool for Windows. Available at: <http://www.putty.org>.
- [15] SQL, Structured Querying Language. Available at: <http://www.w3schools.com/sql/>.
- [16] SSH Communications Security. Available at: <http://www.ssh.com>.

- [17] The FreeBSD Project . Available at: <https://www.freebsd.org>.
- [18] The Unix System. Available at: <http://www.unix.org>.
- [19] XML - Extensible Markup Language. Available at: <http://www.w3.org/XML/>.
- [20] XML-RPC Specification, 1999. Available at: <http://www.xmlrpc.scripting.com/spec>.
- [21] Marta Carbone and Luigi Rizzo. Dummynet revisited. *ACM SIGCOMM Computer Communication Review*, 40(2):12–20, 2010.
- [22] Marta Carbone and Luigi Rizzo. An emulation tool for planetlab. *Computer communications*, 34(16):1980–1990, 2011.
- [23] Roman Chertov, Joseph Kim, and Jiayu Chen. Lte emulation over wired ethernet. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, pages 18–32. Springer, 2012.
- [24] ITU Broadband Commission. The State of Broadband 2012: Achieving Digital Inclusion for All, ITU Broadband Commission Report. Available at: <http://www.broadbandcommission.org/Documents/bb-annualreport2012.pdf>.
- [25] Ericsson. More than 50 Billion Connected Devices, White Paper. Available at: <http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf>.
- [26] Serge Fdida, Timur Friedman, and Thierry Parmentelat. Onelab: An open federated facility for experimentally driven future internet research. In *New Network Architectures*, pages 141–152. Springer, 2010.
- [27] USC Information Sciences Institute. Transmission Control Protocol, DARPA Internet Program, RFC: 793, IETF. Available at: <http://www.tools.ietf.org/html/rfc793>.
- [28] Nadir Javed and Bilhanan Silverajan. Connectivity emulation testbed for iot devices and networks. In *TridentCom 2014, LNICST 137 proceedings*. Springer, 2014.
- [29] Roger P Karrer, Istvan Matyasovszki, Alessio Botta, and Antonio Pescapé. Magnets-experiences from deploying a joint research-operational next-generation wireless access network testbed. In *Testbeds and Research Infrastructure for the Development of Networks and Communities, 2007. TridentCom 2007. 3rd International Conference*, pages 1–10. IEEE, 2007.

- [30] Hermann Kopetz. Internet of things. In *Real-Time Systems*, pages 307–323. Springer, 2011.
- [31] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *ACM SIGCOMM Computer Communication Review*, 33(1):59–64, 2003.
- [32] Larry Peterson and Timothy Roscoe. The design principles of planetlab. *ACM SIGOPS Operating Systems Review*, 40(1):11–16, 2006.
- [33] J. Postel. User Datagram Protocol, RFC: 768, IETF. Available at: <http://www.tools.ietf.org/html/rfc768>.
- [34] ABI Research Press release. The Internet of Things Will Drive Wireless Connected Devices to 40.9 Billion in 2020. Available at: <https://www.abiresearch.com/press/the-internet-of-things-will-drive-wireless-connect>.
- [35] Luis Sanchez, Jose A Galache, Verónica Gutiérrez, Jose M Hernández, Jesús Bernat, Alex Gluhak, and Tomás García. Smartsantander: The meeting point between future internet research and experimentation and the smart cities. In *Future Network & Mobile Summit (FutureNetw)*, 2011, pages 1–8. IEEE, 2011.
- [36] Cisco Systems. The Internet of Things, how the next evolution of the internet is changing everything, White Paper. Available at: http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.
- [37] Hajime Tazaki and Hitoshi Asaeda. Dnemu: Design and implementation of distributed network emulation for smooth experimentation control. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, pages 162–177. Springer, 2012.
- [38] Sebastian Wahle, Bogdan Harjoc, Konrad Campowsky, and Thomas Magedanz. Pan-european testbed and experimental facility federation—architecture refinement and implementation. *International Journal of Communication Networks and Distributed Systems*, 5(1):67–87, 2010.